

Toshiba Global Commerce Solutions
4690 OS



Communications Programming Reference

Version 6 Release 4

Toshiba Global Commerce Solutions
4690 OS



Communications Programming Reference

Version 6 Release 4

Note

Before using this information and the product it supports, be sure to read Safety Information- Read This First, Warranty Information, Uninterruptible Power Supply Information and the information under Appendix F, "Notices," on page 393.

September 2013

This edition applies to Version 6 Release 4 of the licensed program 4690 OS (program number 5639-P70) and to all subsequent releases and modifications until otherwise indicated in new editions.

If you send information to Toshiba Global Commerce Solutions (Toshiba), you grant Toshiba a nonexclusive right to use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright Toshiba Global Commerce Solutions, Inc. 2013.

© Copyright IBM Corporation 2010, 2012

Contents

Figures	xi
Tables	xiii
Safety	xv
About this guide	xvii
Who should use this guide	xvii
Terminal models	xvii
Where to find more information	xviii
4690 V6 Library	xviii
Notice statements	xviii

Part 1. 4690 Communications Services 1

Chapter 1. Introduction to 4690 communications	3
SNA support	3
SNA Communications	3
Link Protocols	4
Link and Session Considerations	5
4690 Non-SNA Support	6
Non-SNA Communications	6
Chapter 2. Using TCP/IP in the operating system	7
4690 TCP/IP file naming conventions	8
Configuring 4690 TCP/IP	10
Enhanced loopback address configuration using ifconfig	12
Using asynchronous communication	12
Identifying a network router	13
Using TCP/IP Host Names	13
resolv File	13
Hosts File	13
Configuring TCP Keepalive and Connection Establishment Timer Values	14
Setting Up and Using Other TCP/IP Applications	14
INETD Superserver (ADXHSI9L.286)	15
FTP Client (ADXHSIGL.286)	15
FTP Server (ADXHSIFL.286)	16
NFS server (ADXHSINL.386)	18
NFS client (ADXHSIDL.286)	19
Telnet client (ADXHSIVL.286)	20
BOOTP client (ADXHSIBL.286)	20
BOOTP server (ADXHSIAL.286)	21
Configuring the PXE Environment and DHCP Server (DHCPD.386)	22
Using TCC and PXE in a routed environment	30
TFTP and MTFTP servers	31
Using the Remote Terminal Utility	31
SNMP agent (ADXHSI1L.286)	33
Telnet server	35
Enhanced Telnet server (ADXHSIUL.286)	36
LPR client (ADXHSIRL.286)	38
Rexec client (ADXHSIXL.286)	39
PCNFSD authentication server	39
4690 TCP/IP programming libraries	40

16-bit to 32-bit conversion	40
Retrieving the local host name	41
TCP/IP in the terminal	41
Chapter 3. Using IP Security Protocol	43
IPsec overview	43
IP Security and the Operating System	43
Enabling IPsec	44
IP Security features	44
IP Security limitations	44
Tunnels and filters.	44
Tunnels and security associations	45
Filtering capability.	46
Configuring manual IPsec tunnels	46
Host-firewall-host configuration	49
Configuring filters	49
User-defined filter rules	49
Auto-generated filter rules	49
Predefined filter rules	50
Direction	50
Subnet masks	50
Examples of filter rules	50
Filters and intrusion detection and prevention.	52
IPsec commands	55
genfilt command (adxipsgf.386)	55
actfilt command (adxipsaf.386)	58
chfilt command (adxipscf.386)	58
mvfilt command (adxipsmf.386)	61
rmfilt command (adxipsrf.386)	61
lsfilt command (adxipslf.386)	62
gentun command (adxipsgt.386)	62
acttun command (adxipsat.386)	65
chtun command (adxipsct.386)	66
rmtun command (adxipsrt.386)	68
lstun command (adxipslt.386)	68
ipsecstat command (adxipsst.386)	69
Chapter 4. Using Secure Shell in the Operating System	71
Introduction to Secure Shell	71
SSH commands	71
Secure Shell Server (ADXSSHD.386)	71
Troubleshooting the SSH Server	73
Secure Shell Client (ADXSSHCL.386)	73
Secure FTP server (ADXSSHPL.386)	76
Secure FTP console session (ADXSSHFL.386)	77
SSH key generator (ADXSSHKL.386)	79
Syntax	80
Command line options	80
Key files	81
Configuration files	81
SSH server configuration file (ADXSSHDF.DAT)	81
SSH client configuration file (ADXSSHCF.DAT)	83
SFTP permissions file (ADXSSHXH.DAT)	84
Authorizing users for SSH or SFTP	85
Running ADXSSHFL (SFTP) as a background application	85
Setup the 4690 OS SSH/SFTP Client	85

Setup the 4690 OS SSH/SFTP Server	87
Chapter 5. Using the Remote Change Management Server	89
The Distributed Systems Executive	89
NetView Distribution Manager	89
RCMS logical unit characteristics	89
RCMS commands.	90
Using the SEND command with a new file.	90
Using the SEND command when the file already exists	90
RETRIEVE command	91
DELETE command	92
INITIATE FUNCTION command	93
INFORM OPERATOR command	93
QUERY command (NetView DM only)	94
RCMS and files.	94
File recovery.	94
Data format file conversion	95
Binary data format	95
EBCDIC data format	95
4690 Programmable Store System data format	95
RCMS translation interface	95
Host access to store controller files	95
Chapter 6. Using the Host Command Processor	97
Introduction to the HCP.	97
Intermittent Transmission of PC-Format Sequential Files	97
Intermittent Transmission of PSS-Format Sequential Files	98
Logical Unit Characteristics	98
HCP Commands	99
ADD KEYED RECORDS Command	99
CREATE FILE Command	100
DELETE KEYED RECORDS Command	102
DUMP FILE Command	102
The HCP EXTENDED DUMP Command	105
LOAD FILE Command	106
PURGE FILE Command	107
READ KEYED RECORD Command	107
REPLACE KEYED RECORDS Command	108
STATUS Command.	108
HCP File Naming Conventions	109
System Files	110
Point-of-Sale Application Files	111
User Files	113
HCP File Security	118
Special Logical Names	118
The HCP Translation Interface.	123
Pipe Interface	123
Message Format.	124
Format Types	125
End of File	126
First Message.	127
Error Handling.	128
Sample Flows From the HCP	129
Chapter 7. Using the Remote Command Processor	131
Introduction to RCP.	131

The RCP Selection File	131
The RCP COMMAND File	132
Command Files on LAN Systems	133
User-Created Output Files	133
Applying Maintenance on a LAN System through RCP	134
The RCP STATUS File	134
RCP Commands	136
Re-IPL Command	136
Remote STC Command	137
Dump Formatter Command	137
Trace Commands	138
Trace Formatter Command	144
File Management Command for Trace Output Data Files	144
System Log Formatter Command	146
File Management Command for System Log	147
Configuration Activation Command	149
Configuration Data Formatter Command	149
Report Module Level Command	150
Audible Alarm Commands	151
Apply Software Maintenance Command	154
Streaming Tape Drive Utility Commands	156
Optical Drive Utility Commands	159
Load All Terminals Command	162
Remote Set Terminal Characteristics Command	163
Set System Time and Date Command	164
File Compression/Decompression Command	165
Problem analysis data compression command	168
Extract Store-Specific Configuration Command	170
Insert Store-Specific Configuration Command	171
Controller BIOS Update Command:	172
Starting RCP	172
Starting RCP from the Host Processor	172
Starting RCP from an Application	172
Starting Applications Using RCP	172
Starting Applications on the 4690 Store System	173
Command Line Applications	173
Background Applications	174
Retrieving Reports Formatted by RCP	177
File Names for Report Files	178
Applying Software Maintenance from a Host Site	178
 Chapter 8. Using Communications and Systems Management	 181
System Alerts	181
User-Application Alerts	181
Vital Product Data File	183
MAC Address Data File	184

Part 2. 4690 Communication Programming 185

Chapter 9. Designing an Application for 3270 Emulation	187
Using the Application Programming Interface	187
Session Identifiers	188
3270 API	188
Starting 3270 Emulation	188
Starting 3270 Emulation from the Command Line (Store Controller Only)	189
Starting 3270 from the API (store controller only)	189

Using a 4680 BASIC interface to access 3270 emulation	189
API verb timeouts	189
API verbs	190
Writing application programs for use with the 3270 API.	201
Compiling and linking	202
Keyboard and language combinations	203
ASCII-EBCDIC translation functions.	204
NLS file structure and contents	204
How to Reconfigure the NLS File.	206
Chapter 10. Designing LU 0/SNA Programs	209
Link and Session Considerations	209
Line Communications Protocol.	209
SNA Support for LU 0	209
Transmission Control Support	209
Data Flow Control Support	209
BIND Processing.	209
Starting LU 0 Applications	210
Application Started by an Unsolicited BIND Request.	210
Opening an SNA Subarea Link	211
Opening an LU 0 Session	211
Closing LU 0 Sessions and Subarea Links	211
4680 BASIC Statements for LU 0 Applications	212
OPEN LINK SNA	212
OPEN SESSION.	212
READ.	213
Example of an SNA Interface with the Host Application.	215
Chapter 11. Designing LU 6.2/SNA Programs	217
CPI Communications Calls	217
4690 Extensions	218
Conditional Disable Link (XCMCDL).	219
Disable Link (XCMDL).	219
Enable Link (XC MEL).	220
Get Last Error (XCMGLE)	220
Query Link Status (XCMQL)	224
Set Timeout Value (XCMSTO).	224
Starting an LU 6.2 TP	225
From an Incoming ALLOCATE Request	225
From the 4690 Main Menu	226
From the Command Line.	226
Establishing an SNA Link from LU 6.2 Applications	226
Ending Links from LU 6.2 Applications	226
Programming Considerations	226
C Language	227
BASIC	227
COBOL	227
Chapter 12. Designing an X.25 Application	229
Hardware Requirements for the X.25 API.	229
Software Requirements for the X.25 API	229
Understanding the X.25 API	229
Hardware Support for the X.25 API	230
Software Support for the X.25 API	230
Implementation of X.25 Protocol Characteristics	231
Principal Features of the X.25 API	231

Summary of Features	232
Error Logging and Recovery	233
System Log Scan	233
Some Common Reasons for Problems	233
Items to Check	233
X.25 Communications Line Trace.	233
Chapter 13. X.25 API Verb Reference	235
CALL Syntax for X.25 API Verbs	235
Supported Verbs	236
XLOAD	236
XOPEN	236
XSEND	239
XRECEIVE	241
XCLOSE.	242
XEVENT.	243
XOPENREC	245
XOPENACC	246
XIT.	247
XRESET.	248
XALLOC.	249
XDEALLOC	250
XSTO.	251
XWAIT (4680 BASIC Only, Multiple Event Wait)	251
ADX_CWAIT (C and COBOL Only, Multiple Event Wait)	252
X.25 API Message Flow	253
Loading the X.25 Driver	253
X.25 API Message Flow — SVC Call Connection	253
X.25 API Message Flow — SVC Call Clearing	254
X.25 API Message Flow — PVC Initialization	256
X.25 API Message Flow — PVC Termination	256
X.25 API Message Flow — Data Transfer	257
X.25 API Message Flow — X.25 Events	258
Remote Loading of Applications	258
X.25 Return Code and Verb Cross-Reference Table	259
Chapter 14. Designing an Asynchronous Program	263
ASYNC Record and Character Support	263
ASYNC XON/XOFF Support	263
ASYNC RS-422 Lines	264
Starting an ASYNC Line	264
Reading Data from the Host	265
Writing Data to the Host	266
Checking the Completion of Read and Write Functions.	266
Controlling Use of Read and Write Buffers	266
Checking and Controlling Modem Interface Signals	267
Example of an ASYNC Application in 4680 BASIC	268
Stopping an ASYNC Line	268
Chapter 15. Designing Applications with C and COBOL Languages	269
Open Communications Link.	269
16-Bit C Interface	269
32-Bit C Interface	269
COBOL Interface.	269
Open SNA Session	270
16-Bit C Interface	270

32-Bit C Interface	270
COBOL Interface.	270
Close Communications Link or Session	271
16-Bit C Interface	271
32-Bit C Interface	271
COBOL Interface.	271
Read Data from Link or Session	271
16-Bit C Interface	271
32-Bit C Interface	271
COBOL Interface.	271
Write Data to Link or Session	272
16-Bit C Interface	272
32-Bit C Interface	272
COBOL Interface.	272
Obtain Status from Link or Session	273
16-Bit C Interface	273
32-Bit C Interface	273
COBOL Interface.	273
Requests to the Driver	276
16-Bit C Interface	276
32-Bit C Interface	276
COBOL Interface.	276
Appendix A. Using HCP with SNA Request/Response Units	279
SNA TH/RH/RU Contents	279
Transmission Header	279
Request/Response Header	279
TH and RH Profiles for Command Requests	280
HCP BIND Request Unit Format	283
Appendix B. HCP Communications with SDLC Protocols	287
Host Communications using SDLC/SNA Communications Protocols	287
Open Series	287
Close Series	287
Load Series	287
Dump Series	289
Action Series	290
Appendix C. Examples of X.25 API Programs and Transaction Programs	293
X.25 API Program Examples and Include Files.	293
XNFBRCV.B86 BASIC File Transfer Receive Program for the X.25 API	293
XNFBSND.B86 BASIC File Transfer Send Program for the X.25 API.	306
Include Files for 4680 BASIC X.25 Application Programs	318
XNFCRCV.HIC File Transfer Receive C Program for the X.25 API	324
XNFCSND.HIC File Transfer Send C Program for the X.25 API	331
Include Files for C X.25 Application Programs	340
Compiling and Linking the X.25 API Programs	345
Starting the Requester for X.25 Programs	346
Starting the Server for X.25 Programs	346
Appendix D. X.25 API Reference Information	347
X.25 Verb Return Codes	347
X.25 Communications Return Codes	351
X.25 Packet Type Identifier	352
X.25 Constants	353
X.25 Event Types	353

X.25 Cause Codes	354
X.25 Network-Generated Diagnostic Codes	355
DTE-Generated Diagnostic Codes	356
DTE Addressing	356
LAPB Commands and Responses - Link Level.	358
Appendix E. Examples of LU 6.2 SNA Transaction Programs	359
C Language TP	359
Example Header File Definitions for Transaction Programs	359
Example CPI for Communications File Requester Program	361
BASIC TPs	370
COBOL TPs	384
Appendix F. Notices	393
Telecommunication regulatory statement	394
Electronic Emission Notices.	394
Federal Communications Commission (FCC) Statement	394
Industry Canada Class A Emission Compliance statement	394
Avis de conformité à la réglementation d'Industrie Canada	394
Australia and New Zealand Class A Statement.	394
European Union Electromagnetic Compatibility (EMC) Directive	
Conformance Statement	394
Germany Class A Statement	395
Japan Voluntary Control Council for Interference Class A statement	396
Japan Electronics and Information Technology Industries Association (JEITA)	
statement	396
Korean communications statement	396
Russian Electromagnetic Interference (EMI) Class A statement.	396
People's Republic of China Class A electronic emission Statement	397
Taiwan Class A compliance statement	397
European Community (EC) Mark of Conformity Statement	398
Electrostatic Discharge (ESD)	398
Japanese Electrical Appliance and Material Safety Law statement	398
Japanese power line harmonics compliance statement.	398
Cable ferrite requirement.	398
Product recycling and disposal	399
Battery return program	400
For Taiwan:.	400
For the European Union:.	400
For California:.	401
Flat panel displays	401
Monitors and workstations	401
Trademarks.	402
Glossary	403
Index	423

Figures

1. Sample DHCP Configuration File	24
2. IP Security - Incoming Packet	45
3. IP Security - Virtual Tunnel	45
4. File Data Received from the HCP.	98
5. Subdirectory File List Description	103
6. Record Formats at the Store Controller and at the Host	126
7. Information That Flows Between the HCP and the Translation Application	127
8. Layout of Format Information for the First Message.	128
9. API Using Network Layer Services	230
10. Addition of X.25 API Applications	232
11. General Syntax of API Verbs	235
12. Confirmation of Local/Remote Delivery of a Data Packet.	242
13. XALLOC Verb Example	250
14. Loading the X.25 Driver for X.25 API Use Only	253
15. Making an Outgoing Call	253
16. Accepting an Incoming Call	254
17. Accepting an Incoming Call with Application Name	254
18. Clearing an Active Circuit	254
19. Clearing an Already Cleared Circuit	255
20. Clearing Call when no XOPENREC waiting	255
21. Initializing a PVC Circuit.	256
22. Terminating Application Use of a PVC Circuit	256
23. Sending Data	257
24. Receiving Data	257
25. Receiving X.25 Events	258
26. TH Profiles Used in Host Processor to Store Controller Communications.	279
27. RH Profiles Used in Host Processor to Store Controller Communications.	280
28. Command Requests from the Host Program	280
29. Command Responses to the Host Program	282
30. Command Requests to the Host Program	283
31. X.25 Packet Type Identifier.	352
32. X.25 Constants	353
33. Event Types	353
34. X.25 Cause Codes	354
35. X.25 Network-Generated Diagnostic Codes: Part A	355
36. X.25 Network-Generated Diagnostic Codes: Part B.	355
37. DTE-Generated Diagnostic Codes	356
38. DTE Addressing Network User Address	357
39. DTE Addressing.	357
40. LAPB Commands and Responses	358

Tables

1. SNA Support	3
2. Ethernet SNA Tunable Parameters	4
3. Absolute Timer Values for Ti, T1, and T2	5
4. 4690 Non-SNA Support	6
5. 4690 TCP/IP memory requirements	7
6. 4690 TCP/IP File Name Index	8
7. DHCP server error message event numbers	29
8. Explanation of filter rules example	51
9. Supported authentication and encryption algorithms	65
10. SFTP interactive commands supported by the 4690 OS SFTP server	76
11. SFTP interactive commands supported by ADXSSHFL386	78
12. Directory information for root directories	91
13. Directory information for system directories	92
14. Directory information for application directories	92
15. Directory information for user directories	92
16. ADD KEYED RECORDS Command Format	99
17. CREATE FILE Command Format	100
18. DELETE KEYED RECORDS Command Format	102
19. Non-LAN System Bytes	103
20. LAN System Bytes	104
21. Returned Data File Attributes	104
22. DUMP FILE Command Format	105
23. EXTENDED DUMP Command Format	106
24. LOAD FILE Command Format	106
25. PURGE FILE Command Format	107
26. READ KEYED RECORD Command Format	107
27. REPLACE KEYED RECORDS Command Format	108
28. STATUS Command Format	108
29. Status Response Format	109
30. Subdirectory (Dump of List of Files in the Named Subdirectory)	110
31. Naming System Files	110
32. Naming Point-of-Sale Application Files	111
33. Naming User Files	113
34. File Use Table	114
35. Translation Table	114
36. ASCII-EBCDIC Translation Table for Print Files	115
37. HCP File Security	118
38. Special Logical Names	118
39. 6-Character Names That Do Not Follow ADX Naming Conventions	123
40. Record Header for HCP Translation Interface	124
41. Interface Dialogues	130
42. RCP STATUS File Entries	134
43. Reports Formatted by RCP	177
44. HCP File Names for Report Files Used with RCP	178
45. Alert Numbers for Master Controller Event Numbers	181
46. Alert Numbers for Subordinate Controller Event Numbers	182
47. Alert Numbers for Terminal Event Numbers	182
48. API verb timeouts	189
49. Valid message numbers for OGL messages	196
50. KEY.CODE% Values for 3270 function keystrokes	197
51. Device control key codes	198
52. National Language Support for 3270 emulation	203
53. National Language Support for countries without 3270 emulation files	204

54.	Information Messages Displayed by 3270 Emulation Program	206
55.	CPI Communications Calls.	217
56.	Overview of 4690 Extension Routines.	218
57.	Variable Types and Lengths	219
58.	verb_id Values of CPI Communications Calls	221
59.	verb_id Values of 4690 Extensions	222
60.	XCMGLE Error Information Using BASIC	224
61.	User Data Field Priority	239
62.	X.25 Verb Return Codes Cross-Reference Table.	260
63.	Required Active Modem Control Lines for OPEN, TRANSMIT, and RECEIVE	265
64.	Status Information Represented by Byte SS	266
65.	ASYNCR Status Flags for Byte 4	273
66.	ASYNCR Status Flags for Byte 5	274
67.	ASYNCR Status Flags for Byte 6	274
68.	ASYNCR Status Flags for Byte 7	274
69.	SNA Status Flags for Byte 5	274
70.	SNA Status Flags for Byte 6	275
71.	SNA Status Flags for Byte 7	275
72.	SNA Link Flags for Byte 4	275
73.	SNA Link Flags for Byte 5	276
74.	ASYNCR Status Flags for Byte 4	277
75.	SNA Status Flags for Byte 4	277
76.	Formats of the HCP BIND Request Unit	283
77.	Variables Used by External Functions.	294
78.	String Variables Used in This Program	295
79.	Integer*2 Variables Used in This Program	296
80.	Integer*4 Variables Used in This Program	298
81.	Variables Used by External Functions.	307
82.	String Variables Used in This Program	309
83.	Integer*2 Variables Used in This Program	310
84.	Integer*4 Variables Used in This Program	311
85.	Verb Structure Variables.	324
86.	Variables Used in This Program	325
87.	Verb Structure Variables.	332
88.	Variables Used in This Program	334
89.	X.25 Verb Return Codes and Descriptions	347
90.	X.25 Communications Return Codes	351

Safety

Before installing this product, read Safety Information- Read This First.

قبل تركيب هذا المنتج، يجب قراءة الملاحظات الأمنية

Antes de instalar este produto, leia as Informações de Segurança.

在安裝本產品之前，請仔細閱讀 **Safety Information**
(安全信息)。

安裝本產品之前，請先閱讀「安全資訊」。

Prije instalacije ovog produkta obavezno pročitajte Sigurnosne Upute.

Před instalací tohoto produktu si přečtěte příručku bezpečnostních instrukcí.

Læs sikkerhedsforskrifterne, før du installerer dette produkt.

Lees voordat u dit product installeert eerst de veiligheidsvoorschriften.

Ennen kuin asennat tämän tuotteen, lue turvaohjeet kohdasta Safety Information.

Avant d'installer ce produit, lisez les consignes de sécurité.

Vor der Installation dieses Produkts die Sicherheitshinweise lesen.

Πριν εγκαταστήσετε το προϊόν αυτό, διαβάστε τις πληροφορίες ασφάλειας
(safety information).

לפני שתתקינו מוצר זה, קראו את הוראות הבטיחות.

A termék telepítése előtt olvassa el a Biztonsági előírásokat!

Prima di installare questo prodotto, leggere le Informazioni sulla Sicurezza.

製品の設置の前に、安全情報をお読みください。

본 제품을 설치하기 전에 안전 정보를 읽으십시오.

Пред да се инсталира овој продукт, прочитајте информацијата за безбедност.

Les sikkerhetsinformasjonen (Safety Information) før du installerer dette produktet.

Przed zainstalowaniem tego produktu, należy zapoznać się
z książką "Informacje dotyczące bezpieczeństwa" (Safety Information).

Antes de instalar este produto, leia as Informações sobre Segurança.

Перед установкой продукта прочтите инструкции по
технике безопасности.

Pred inštaláciou tohto zariadenia si pečítajte Bezpečnostné predpisy.

Pred namestitvijo tega proizvoda preberite Varnostne informacije.

Antes de instalar este producto, lea la información de seguridad.

Läs säkerhetsinformationen innan du installerar den här produkten.

About this guide

This guide describes procedures for customizing communications programs for use with the 4690 Store System. It is intended to help programmers develop communications for 4690 OS Version 6 Release 4 (hereafter called the *operating system*) to meet the specific needs of their enterprise. Beginning with 4690 OS Version 6 Release 4, Token Ring communications is no longer supported.

- | As of September 28, 2013, SNA communications will no longer be supported on any 4690 OS releases.

Who should use this guide

This guide is intended for programmers who are familiar with the concepts and facilities of the operating system and the 4690 Store System. This guide assumes that you are familiar with the following concepts and facilities:

- Asynchronous communications protocols
- Local area networks (LANs)
- Systems Network Architecture (SNA)
- TCP/IP
- X.25 networks

Terminal models

- | The 4693-xx1/4694, SurePOS™ 300/700 Series and TCxWave 6140 Series terminals are called *Mod1*
- | terminals. Although all are called Mod1 terminals, each terminal model supports some features that other
- | models do not support. Additionally, the SurePOS 300/700 Series and TCxWave 6140 Series terminals
- | provide Universal Serial Bus (USB) capabilities.

The 4693-xx2 terminals are called *Mod2* terminals. These terminals attach to a Mod1 terminal and depend upon that Mod1 terminal for control and communication with the store controller.

Note: 4683 terminals are not supported on 4690 OS V6R3 or later. References to 4683 terminals only apply to previous versions of the OS.

The controller/terminal (for example, a 4693-5x1 controller/terminal) combines the function of the store controller and point-of-sale terminal in a single product. The terminal portion of a controller/terminal is considered to be a *Mod1* terminal.

Note: The 4694 and SurePOS 700 Series (except models Cxx) systems are valid as controller or terminals. The 4693 systems are only supported as a controller in a non-Java™ environment or as an alternate in a Java environment.

4690 V6R3 introduced support for the SurePOS 300 Series Model 350 terminal (4810-350). This is a *Mod1* terminal. The 4810-350 has the following characteristics:

- It can only be used as a terminal in Enhanced Mode
- It cannot be used as a store controller
- The RS232 Sureport card is not supported by 4690 OS
- There are no RS485 ports available

4690 V6R4 introduced support for the TCxWave Series terminal (6140-100). The 6140-100 is a *Mod1* terminal with the following characteristics:

- It can only be used as a terminal in Enhanced Mode
- It cannot be used as a store controller
- No RS485 ports available, including no cash drawer port

- Cash drawer support is available through USB-attached cash drawers
- RS232 support is provided through RS232-to-Serial dongles
- The Power Button functions as the only available Dump Button

Where to find more information

Current versions of Toshiba publications are available on the Toshiba support site.

1. On the right side of the web page under popular links, select **Publications**.
2. Click on the publication related to your product.

4690 V6 Library

4690 OS Version 6: Planning, Installation, and Configuration Guide, G362-0541

4690 OS Version 6: User's Guide, G362-0542

4690 OS Version 6: Messages Guide, G362-0543

4690 OS Version 6: Communications Programming Reference, G362-0544

4690 OS Version 6: Programming Guide, G362-0545

4690 OS Version 6: Master Index, G362-0546

4680 BASIC: Language Reference, SC30-3356

Note: References to related 4690 publications in this guide are reference to the publications in the 4690 Version 6 library.

Notice statements

Notices in this guide are defined as follows:

Notes	These notices provide important tips, guidance, or advice.
Important	These notices provide information or advice that might help you avoid inconvenient or problem situations.
Attention	These notices indicate potential damage to programs, devices, or data. An attention notice is placed just before the instruction or situation in which damage could occur.
CAUTION	These statements indicate situations that can be potentially hazardous to you. A caution statement is placed just before the description of a potentially hazardous procedure step or situation.
DANGER	These statements indicate situations that can be potentially lethal or extremely hazardous to you. A danger statement is placed just before the description of a potentially lethal or extremely hazardous procedure step or situation.

Part 1. 4690 Communications Services

Chapter 1. Introduction to 4690 communications

This guide assumes that you are familiar with communications concepts and functions. Knowledge of communication protocols is vital in understanding this chapter and this guide.

SNA support

Note: As of September 28, 2013, SNA communications will no longer be supported on any 4690 OS releases.

Table 1 summarizes Systems Network Architecture (SNA) support.

Table 1. SNA Support

Session Protocols	Link protocols	Languages		
	SDLC	X.25	C	BASIC
3270	X	X		X
LU 0	X	X	X	X
LU 6.2	X	X*	X	X

* Single-session dependent only

SNA Communications

Type 2.0 Node

The 4690 store controller can connect to an SNA subarea network as peripheral or a type 2.0 node. As a type 2.0 node, the 4690 store controller depends on a host to perform the subarea node role to activate network resources and route data and messages. A type 2.0 node contains one physical unit (PU) and one or more logical units (LU) as network resources. The PU participates with the host to control links attached to the node and the LUs. The LUs provide end-user program access to the network. Using a *session*, an SNA program at the 4690 store controller can communicate with a partner SNA program at the host. Depending on the message format and the protocol supported by the LUs involved, there can be many types of sessions.

The SNA support provides for session type 0 using a programming interface for LU 0, session types 1, 2 and 3 using a programming interface for 3270 emulation, and session type 6.2 using the CPI for communications. It can support logical unit (LU) 0 applications, 3270 emulation, and LU 6.2 transaction programs.

Every type 2.0 node contains one physical unit (PU) to manage and monitor the attached links and other resources in response to requests by the system services control point (SSCP) of a host (node type 5). The physical unit connects the node to adjacent nodes. It represents the processor, the store controller, the workstation, or the printer to the network.

A type 2.0 node supports subarea links for connection with a host processor on LU 0, 3270 emulation, and LU 6.2.

Type 2.1 Node

The 4690 store controller can also connect to an SNA network as a type 2.1 node. It performs similar to a type 2.0 node in the way it supports LUs and session types. It supports dependent LU types, such as LU 0 applications and 3270 emulation, and independent LUs (LU 6.2 applications). You can use Synchronous Data Link Control (SDLC) line connections to support both independent and dependent LUs.

The 4690 store controller can connect to a peer processor as an SNA type 2.1 node. It can support only LU 6.2 transaction programs. You can use SDLC and line connections on a peer connection. A type 2.1 node also supports logical connection that enables two transaction programs to communicate in the same store controller using LU 6.2 protocols. This logical connection is called the *local link*.

A type 2.1 node supports peer links or subarea links (XID 3) on LU 6.2. A peripheral node control point (PNCP) activates sessions with PUs and LUs that are located in type 2.1 nodes. A PNCP is a control point in a type 2.1 node that provides a subset of the SSCP functions. A PNCP can activate, deactivate, and manage network resources in type 2.1 nodes. Like SSCPs, a PNCP becomes active when its type 2.1 node becomes operational.

Link Protocols

SDLC

The SDLC/SNA link protocol supports secondary or negotiable mode operation with type 2.0 and 2.1 nodes. Depending on the SDLC link type, the station mode operates as follows:

Link Type	Station Mode
Subarea (XID 0)	Secondary
Subarea (XID 3)	Negotiable
Peer	Negotiable

Ethernet

The Ethernet/SNA link protocol supports type 2.0 and 2.1 nodes. The Ethernet supports locally- and universally-administered node addresses.

Ethernet SNA Tunable Parameters: You can set 11 logical names to override certain Ethernet data-link-control parameters for SNA on the Ethernet. These control parameters should only be changed by personnel with Ethernet LAN expertise. The names and the valid range for the definition of the logical names are described in Table 2.

Table 2. Ethernet SNA Tunable Parameters

Logical Name	Meaning	Valid range
ADXSNAFL	SNA Flags	Hexadecimal values describing SNA flags
ADXSNATI	Inactivity Timer (Ti)	1 – 10
ADXSNA1	Response Timer (T1)	1 – 10
ADXSNA2	Receive Ack Timer (T2)	1 – 10
ADXSNA2	Max Retry Count (N2)	1 – 255 (default 8)
ADXSNA3	Max In (N3)	1 – 127 (default 1)
ADXSNAW	Max Out Increment (NW)	1 – 255 (default 1)
ADXSNAW	Max Out (TW)	1 – 127 (default 2)
ADXSNAI	Retry Timeout	5 – 120 seconds (default 5)

All of the data link control parameters for Ethernet have defaults that are used if the logical name is not defined, or if it is defined to an illegal value.

Timer values Ti, T1, and T2 can be set to any value between 1 and 10 using logical names ADXSNATI, ADXSNA1, and ADXSNA2, respectively. This represents a relative timer setting. The absolute timer value (approximately) can be correlated as shown in Table 3 on page 5.

Table 3. Absolute Timer Values for T_i , T_1 , and T_2

T_i		T_1		T_2	
set	absolute	set	absolute	set	absolute
1	1 second	1	200 ms	1	40 ms
2	2 seconds	2	400 ms	2	80 ms
3	3 seconds	3	600 ms	3	120 ms
4	4 seconds	4	800 ms	4	160 ms
5	5 seconds	5	1 second	5	200 ms
6	10 seconds	6	2 seconds	6	400 ms
7	20 seconds	7	4 seconds	7	800 ms
8	30 seconds	8	6 seconds	8	1.2 seconds
9	40 seconds	9	8 seconds	9	1.6 seconds
10	50 seconds	10	10 seconds	10	2 seconds

The value for ADXSNAXI is used to control the retry timeout associated with XIDs (an XID is an initial station contact message) transmitted across a Ethernet bridge when attempting to contact a partner station that is not attached to the local Ethernet LAN segment. This value can be used to adjust how often (in seconds) the XID is retried across the bridge. The default value is 5 seconds.

The value for ADXSNALX is used to control the number of XIDs transmitted on the local LAN segment when attempting to contact a partner station. When SNA is activated, XIDs are initially sent on the local LAN segment. If no response is received from a partner station, the XID is then transmitted to bridged LAN segments. ADXSNALX is a count that controls how many times the XID is transmitted on the local LAN segment before it is subsequently transmitted to the bridged LAN segments. The default is 12 retries.

X.25

The X.25 link protocol supports type 2.0 nodes for communication with an SNA type 5 node. The store controller functions as a secondary node using the qualified logical link control protocol (QLLC), and supports LU 0, 1, 2, 3, and LU 6.2 SNA protocols.

The store controller can attach to an X.25 network by an X.25 Interface Co-Processor/2 Adapter. You can use up to four QLLC virtual circuits on each X.25 Interface Co-Processor/2 adapter. The operating system allows use of X.25 communication protocols defined in the 1980, 1984, or 1988 International Telegraph and Telephone Consultative Committee (CCITT) recommendations.

Link and Session Considerations

The operating system can support one SDLC/SNA link for each SDLC communications port and up to four X.25/SNA links for each Realtime Interface Co-Processor Multiport/2 Adapter. All links might be active concurrently or only a subset might be active, depending upon restrictions for the particular adapters installed on the store controller.

A maximum of 100 LU 0 and 3270 sessions can be active across all links. A maximum of 15 LU 0 sessions can be active on one link. The limit for dependent sessions (LU 0 and 3270) on a single link is 47. Also, no limit exists on the number of LU 6.2 sessions that can be active, except as governed by memory availability. A maximum of 32 conversations can be active in all LU 6.2 sessions.

4690 Non-SNA Support

Table 4 summarizes 4690 non-SNA support.

Table 4. 4690 Non-SNA Support

APIs	Languages		
	C	BASIC	COBOL
X.25	X	X	X
ASYNCR	X	X	X

Non-SNA Communications

X.25

The operating system provides an application programming interface (API) for user-written programs. For more information about the X.25 APIs, see Chapter 12, “Designing an X.25 Application,” on page 229 or Chapter 13, “X.25 API Verb Reference,” on page 235.

ASYNCR

The user application must provide and interpret any control characters within the transmitted data. If the selection is made during configuration of the operating system, the ASYNCR support can only provide the following options:

- XON/XOFF processing
- End-of-record (EOR) character control

See Chapter 14, “Designing an Asynchronous Program” for more information on ASYNCR support. See the *4690 OS: Planning, Installation, and Configuration Guide* for the configuration options that you can use with ASYNCR.

04/07804/

Chapter 2. Using TCP/IP in the operating system

This chapter describes using TCP/IP in both store controllers and terminals. Configuring the PXE environment and configuring a DHCP server are discussed in this chapter. The chapter assumes you are familiar with TCP/IP and, therefore, describes only how TCP/IP is used with the operating system. If you need detailed information about TCP/IP, see the *Transmission Control Protocol/Internet Protocol for 4690 Application Interface Guide*, which can be found on the Toshiba support site. Select **Publications** to find information on the following items.

TCP/IP support within the operating system is implemented as a communications device driver, similar to the SNA driver and the NetBIOS driver. 4690 TCP/IP can operate concurrently with NetBIOS, Terminal Controller Communications (TCC), and SNA protocols.

4690 TCP/IP is also supported on the Ethernet network. All Ethernet adapters supported by the operating system are supported by 4690 TCP/IP. All 4690 TCP/IP functions and applications are available on the Ethernet LAN.

User-level access to the driver is provided through the socket programming interface. A socket runtime library is provided that can be linked to user-written applications. In addition to the TCP/IP driver and socket library, a SNMP agent is implemented.

The 4690 TCP/IP stack does not provide the client or server implementation of the SOCKS proxy protocol. In the case of an internal (enterprise) network within a firewall, TCP/IP applications are not enabled to communicate through that firewall.

4690 TCP/IP requires approximately 3 MB of free hard disk storage. Memory requirements for 4690 TCP/IP vary depending on the number and type of TCP/IP applications that are active. However, at a minimum, approximately 500 KB to 800 KB of RAM should be available. This allows for approximately 400 KB for the TCP/IP driver (code and data), and one or two TCP/IP applications (for example, FTP and Telnet). Table 5 shows specific memory requirements for each function.

Note: TCP/IP driver memory is required as a minimum and all other memory calculations for optional applications should be added to the driver memory.

Table 5. 4690 TCP/IP memory requirements

Function	Memory (KB)
TCP/IP driver	400*
FTP client	220
FTP server	200
TFTP server	200
INETD superserver	50
BOOTP client	120
BOOTP server	200
Telnet client — VT100	200
Telnet server	
ADXHSIIL	375
ADXHSIUL	104
SNMP agent	230
Portmapper	220
NFS server	275

Table 5. 4690 TCP/IP memory requirements (continued)

Function	Memory (KB)
Rexec client	200
LPR client	180
NFS client	200
DHCP server	300

* Driver memory is 400 KB for the default of 48 sockets. The maximum number of sockets is configurable up to 600. Each socket above 48 allocates another 2 KB of memory. To configure more than 48 sockets, see "Configuring 4690 TCP/IP" on page 10.

4690 TCP/IP can operate in one of two 4690 system configurations:

- 4690 LAN environment in which Distributed Data Application (DDA) is being used for store controller communications, file mirroring, and file services. In this environment the 4690 store controller can have a role of Master, File Server, Alternate Master, Alternate File Server, or Subordinate.
- Non-DDA LAN environment in which the store controller operates independent of any other controller on the LAN. Typically, this environment consists of a single store controller connected with other machines using a LAN adapter. It is normally designated as the Master store controller although there are no Alternate or Subordinate store controllers.

When running TCP/IP applications in the controller, it is recommended that the HOSTNAME logical name be defined to the value of the TCP/IP host name of the controller. The Java™ programming language provides a method called `getLocalHost()`, which returns the TCP/IP host name of the machine on which the Java application is running. Invalid results will return from this method if the HOSTNAME logical name is not defined.

4690 TCP/IP file naming conventions

Most 4690 TCP/IP software program files, data files, and programming libraries adhere to the operating system file naming conventions. In some cases, logical names are defined when 4690 TCP/IP is loaded. Table 6 maps the 4690 file names with logical names and their respective function. The logical names attempt to provide the standard UNIX program names for the 4690 TCP/IP files. Wherever a logical name is defined for a physical file name, it can be used in place of the name and the operating system performs the necessary replacement of the physical file name.

Note: The API runtime libraries (*.L86 and .LIB) can be found on the 4690 Optionals. These files are not copied onto the controller during the Apply Software Maintenance function.

Attention: You must not define a logical file name of "tcpip" because this file name is reserved for use by the 4690 TCP/IP driver.

Table 6. 4690 TCP/IP File Name Index

4690 store controller physical file name	4690 logical name	Description	OS/2 TCP/IP file name
C:\ADXHSIDL.L86	none	SNMP DPI library	dpi.lib*
C:\ADXHSIRL.L86	none	Sun RPC library	sunrpc.lib*
C:\ADXHSISL.L86	none	Socket library	tcpip.lib
C:\ADXHSITL.L86	none	FTP API library	ftppapi.lib*
C:\ADX_SDT1\ADXHSIAF.DAT	none	BOOTP Server Bootptab	etc/bootptab
C:\ADX_SDT1\ADXHSIDF.DAT	None	SNMP trap destination	snmptrap.dst

Table 6. 4690 TCP/IP File Name Index (continued)

4690 store controller physical file name	4690 logical name	Description	OS/2 TCP/IP file name
C:\ADX_SDT1\ADXHSIGF.DAT	none	FTP client netrc	etc/netrc
C:\ADX_SDT1\ADXHSIHF.DAT	hosts	Local hosts file	etc/hosts
C:\ADX_SDT1\ADXHSIIF.DAT	none	INETD data	inetd.lst
C:\ADX_SDT1\ADXHSIMF.DAT	none	Telnet server messages	none
C:\ADX_SDT1\ADXHSINF.DAT	networks	Lists networks	etc/networks
C:\ADX_SDT1\ADXHSIPF.DAT	protocol	Lists IP protocols	etc/protocol
C:\ADX_SDT1\ADXHSIQF.DAT	none	SNMP community names	pw.src
C:\ADX_SDT1\ADXHSIRF.DAT	resolv	Identifies name server	etc/resolv
C:\ADX_SDT1\ADXHSISF.DAT	services	Lists servers and ports	etc/services
C:\ADX_SDT1\ADXHSIU.F.DAT	none	FTP server trusers	etc/trusers
C:\ADX_SDT1\ADXHSIXF.DAT	none	NFS exports	etc/exports
C:\ADX_SDT1\ADXHSIZF.DAT	none	SNMP environment names	none
C:\ADX_SDT1\ADXIP00D.DAT	None	Sample DHCP configuration file	none
C:\ADX_SDT1\ADXIP00Z.BAT	None	Sample config.batch	SETUP.CMD
C:\ADX_SDT1\PCNFSD.DAT	none	PC NFS Users	none
C:\ADX_SDT1\RPC.DAT	none	RPC server names	etc/rpc
C:\ADX_SPGM\ADXHSI0L.BSX	none	TCP/IP driver symbols	none
C:\ADX_SPGM\ADXHSI0L.286	none	TCP/IP protocol driver	bin/inet.sys
C:\ADX_SPGM\ADXHSI1L.286	none	SNMP agent	bin/snmpd.exe
C:\ADX_SPGM\ADXHSI2L.286	none	Initialize TCP/IP	bin/tcpstart.exe
C:\ADX_SPGM\ADXHSI3L.286	ifconfig	Configure interface	bin/ifconfig.exe
C:\ADX_SPGM\ADXHSI4L.286	route	Define network rout	bin/route.exe
C:\ADX_SPGM\ADXHSI5L.286	ping	Ping program	bin/ping.exe
C:\ADX_SPGM\ADXHSI6L.286	netstat	Check network status	bin/netstat.exe
C:\ADX_SPGM\ADXHSI7L.286	finger	Check remote users	bin/finger.exe
C:\ADX_SPGM\ADXHSI8L.286	none	SNMP names encryption	bin/make_pw.exe
C:\ADX_SPGM\ADXHSI9L.286	none	INETD superserver	bin/inetd.exe
C:\ADX_SPGM\ADXHSIAL.286	bootpd	Bootp server	bin/bootpd.exe
C:\ADX_SPGM\ADXHSIBL.286	bootp	Bootp client	bin/bootp.exe
C:\ADX_SPGM\ADXHSICL.286	none	Bootp exfile compl	none
C:\ADX_SPGM\ADXHSIFL.286	none	FTP server	bin/ftpd.exe
C:\ADX_SPGM\ADXHSIGL.286	ftp	FTP client	bin/ftp.exe
C:\ADX_SPGM\ADXHSIIL.286	none	Telnet server (keyboard)	bin/telnetd.exe
C:\ADX_SPGM\ADXHSILL.286	none	Lock local keyboard	none
C:\ADX_SPGM\ADXHSINL.386	none	NFS server	bin/nfsd.exe

Table 6. 4690 TCP/IP File Name Index (continued)

4690 store controller physical file name	4690 logical name	Description	OS/2 TCP/IP file name
C:\ADX_SPGM\ADXHSIPL.286	none	RPC Portmapper	bin/portmap.exe
C:\ADX_SPGM\ADXHSIRL.286	lpr	LPR client	bin/lpr.exe
C:\ADX_SPGM\ADXHSISL.286	none	Telnet server (screen)	none
C:\ADX_SPGM\ADXHSITL.286	tftpd	Tftp server	bin/tftpd.exe
C:\ADX_SPGM\ADXHSIUL.286	none	Telnet server (Enhanced)	bin/telnetd.exe
C:\ADX_SPGM\ADXHSIVL.286	vt100	VT100 telnet client	bin/VT100.exe
C:\ADX_SPGM\ADXHSIXL.286	rexec	REXEC client	bin/rexec.exe
C:\ADX_SPGM\DHCPD.386	none	DHCP server file	none
C:\ADX_SPGM\DHCPINFO.386	none	DHCP server utility	none
C:\ADX_SPGM\HOST.286	none	Host name resolution	bin/host.exe
C:\ADX_SPGM\PCNFSD.286	none	PC NFS authenticator	bin/pcnfsd.exe
C:\ADX_SPGM\RPCINFO.286	none	RPC status	bin/rpcinfo.exe
C:\ADX_SPGM\SLIO.286	none	SLIP driver application	bin/slip.exe
C:\ADX_SPGM\TRACERTE.286	none	Packet routing trace	bin/tracerte.exe

* These are 16-bit only.

Configuring 4690 TCP/IP

The 4690 TCP/IP protocol driver is loaded during the IPL of the store controller based on the existence of the ADXIP??Z.BAT configuration batch file in the C:\ADX_SDT1 subdirectory, where ?? is the local controller ID. A sample file, ADXIP00Z.BAT, is copied to the store controller during installation. You must rename or copy this file to the correct name to reflect the ID of your store controller in order to activate TCP/IP. This file has a compound distribution attribute and will be copied to all subordinate controllers. For any subordinate store controllers running TCP/IP, you can create the ADXIP??Z.BAT file for that store controller on the master store controller and distribute the file.

Note: You must have the configuration batch file on each machine that you want to run 4690 TCP/IP.

The configuration batch file must contain the following lines:

```
adxhsi2l [<number of sockets>]
ifconfig lan0 <ip_address>
```

where *ip_address* contains this machine's (host) Internet address in a dotted decimal format. For example 9.67.39.81 is a valid address. The assignment of addresses for a proprietary network are enterprise defined. Addresses for the public Internet must be obtained from the administrators of the Internet. All TCP/IP hosts in the network should have a unique Internet address.

Depending upon your configuration, you might have to use the *netmask* parameter on the **ifconfig** command. For example:

```
ifconfig lan0 <ip_address> netmask 255.255.248.0
```

This parameter tells the TCP/IP stack how to interpret the IP address.

Note: IP addresses can be specified as decimal, hexadecimal, or octal. A leading 0x indicates hex. A leading 0 (zero) indicates octal. Anything else indicates decimal. For example, the following three IP addresses are identical.

- 9.67.39.83
- 0x9.0x43.0x27.0x53
- 011.0103.047.0123

In TCP/IP files that contain IP addresses, any leading zeros in the addresses are interpreted as octal, not decimal. However, in terminal configuration, IP addresses are read as decimal.

If your system is an Enhanced Mode system and will run Java 6 applications which require IP communications, the *elooaddr* parameter may also be required on the **ifconfig** command. Refer to “Enhanced loopback address configuration using ifconfig” on page 12 for more details.

To configure TCP/IP on both adapters (or interfaces), the IP configuration batch file would contain the following lines:

```
adxhsi2l
ifconfig lan0 <lan0 ip address> netmask< subnet mask>
ifconfig lan1 <lan1 ip address> netmask< subnet mask>
```

Notes:

1. The **lan0** and **lan1** addresses must be configured on different IP subnets.
2. The store controller in this configuration has more than one interface and thus, more than one IP address (often referred to as multihomed).
3. When addressing a multihomed controller from a remote system, be sure to use the controller IP address that is reachable on that network. For example, if configuring an NFS client on a terminal to mount a drive on a store controller, the NFS Server IP address of the controller must be configured. If the NFS Server is running on a 4690 multihomed store controller, the **lan1** IP address of the controller is the correct address to use for the NFS Server IP address because it is the IP address on the TCC network.

The logical names, **lan0** and **lan1**, represent the LAN media interface and the TCC media interface, respectively. The logical names, **lan0** and **lan1**, are set to either **en0** which refers to the Ethernet adapter."

After configuring TCP/IP on a 4690 store controller, it is advisable to use the following utilities to ensure that your network is configured properly.

- netstat -a displays the addresses of the configured IP network interfaces.
- ifconfig lan0 and/or ifconfig lan1 displays IP configuration information for the specified interface.
- ping <ip address> sends an echo request to <ip address> to determine if a remote host is accessible. If successful, your TCP/IP network is operational.

The ADXHSI2L command is equivalent to the OS/2® TCPSTART command. However, ADXHSI2L can accept a single parameter that indicates the maximum number of sockets that can be active at any one time. If no parameter is specified, the default is 48 sockets, which is sufficient for most instances. If more sockets are needed, enter a value between 49 and 600 after the ADXHSI2L command. For example, to configure 217 sockets, edit the line to read:

```
adxhsi2l 217
```

Notes:

1. Additional memory is allocated for each socket above 48 (about 2 KB per socket) regardless of whether the socket has an active connection. Also, when set to a value, the maximum number of sockets cannot be changed by a subsequent call of ADXHSI2L. The maximum number of sockets can be set only once after a machine IPL. The maximum sockets value represents the combined total of both UDP and TCP sockets.
2. Controller to Controller Communication over IP (CCC/IP) requires the use of extra sockets. It is recommended that you increase the maximum number of sockets that can be active at one time by 24 on each controller configured for CCC/IP.

3. If you use the Store Integrator Facility, increase the maximum number of sockets that can be active at one time to at least 128 to avoid running out of sockets.

For more information regarding the IFCONFIG command and Internet addressing, see the *Transmission Control Protocol/Internet Protocol for 4690 Application Interface Guide*.

4690 TCP/IP on Ethernet networks can support DIX V2 Ethernet or IEEE 802.2/802.3 Ethernet. The default interface setting is DIX V2. If IEEE 802.2/802.3 is required, you must use the 802.3 command line switch for the IFCONFIG command.

Enhanced loopback address configuration using ifconfig

If IP communications are required between Java 6 applications and other 4690 applications on the same Enhanced Mode system, you must configure an enhanced loopback address. The enhanced loopback address is an IP address on the same subnet as the system running the Java 6 application. A specific IP address may be configured or the keyword "last" may be used to specify the last valid IP address on the subnet.

To configure the enhanced loopback address on the controller, modify the IP configuration file, `adx_sdt1:adxip??z.bat`, and add the `elooaddr` parameter on the `ifconfig` command as shown in the examples below:

```
ifconfig lan0 10.1.1.1 netmask 255.255.255.0 elooaddr 10.1.1.200
```

```
ifconfig lan0 10.1.1.1 netmask 255.255.255.0 elooaddr last (configures the elooaddr as 10.1.1.254)
```

To configure the enhanced loopback address on the terminal, modify the Terminal IP configuration file, `adx_spgm:ipconfig.dat`, and add a new line to configure the `elooaddr` parameter as follows:

```
ifconfig lan0 elooaddr last
```

After changing `adx_spgm:ipconfig.dat`, run `terminal loadshrink (adxrtcl)` and reload the terminal.

To configure the enhanced loopback address on a controller configured for bootp, add an `ifconfig` command to the bootp configuration file, `adx_sdt1:adxbp??z.bat`, with only the `elooaddr` parameter as follows:

```
ifconfig lan0 elooaddr last
```

Using asynchronous communication

With 4690 TCP/IP, you can configure and activate an ASYNC line for SLIP support. See the worksheets in the *4690 OS: Planning, Installation, and Configuration Guide* to configure asynchronous communications. You must reboot the store controller and activate the new line configuration. The following steps tell you how to configure and activate this support.

Note: SLIPLINE is a sample name.

1. From a 4690 command line, enter:

```
SLIO SLIPLINE
```

If you want to start SLIP on your controller after each reboot, you can add this line to your TCP/IP initialization BAT file (`ADX_SDT1:ADXIPxxZ.BAT`, where `xx` is the controller node ID).

2. Configure the TCP/IP SLIP interface with the *ifconfig* command. The interface name is **sl** for 4690 SLIP. For example, `ifconfig sl 129.67.39.90 129.67.39.3 netmask 255.255.255.0`.

The first IP address is the local SLIP interface address. The second IP address is the destination address. The netmask 255.255.255.0 parameter is optional.

3. Enter *ifconfig* from a 4690 command line to receive a usage message for this command.

Identifying a network router

If your network contains an IP router, file ADXIP??Z.BAT can identify it using the ROUTE command. For example:

```
route add default <router_ip_address> 1
```

where *router_ip_address* identifies the Internet address of the network router in a dotted decimal format. The ROUTE command should follow the IFCONFIG command in the configuration batch file ADXIP??Z.BAT.

Using TCP/IP Host Names

TCP/IP allows a host name to be used in place of the Internet address. To resolve a name to an Internet address, these two methods are used:

- TCP/IP contacts the name server host that is designated in the RESOLV file using domain name system (DNS) protocols to resolve the name.
- The local HOSTS file is searched for a match to resolve the name.

The order in which the two name resolution methods are used depends on whether you have defined the ADXDSN2S logical name.

- If ADXDNS2S is not defined, TCP/IP contacts the name server host first. If the name server cannot resolve the name, then the local HOSTS file is searched. This is the only search order supported on 4690 OS prior to Version 4.
- If ADXDNS2S is defined, the local HOSTS file is first searched for a match and, if a match is not found, then the name server is contacted to resolve the name. This search order is only supported on 4690 OS Version 4 or higher.

resolv File

TCP/IP attempts to contact the name server only if the C:\ADX_SDT1\ADXHSIRF.DAT RESOLV file exists and identifies the domain and the address of the name server. During the 4690 TCP/IP installation, a sample RESOLV file is copied onto the 4690 store controller. The domain and nameserver entries in this sample RESOLV file are commented out. If your TCP/IP network uses a name server, you should uncomment both of these entries in the RESOLV file and then change them to match your IP network configuration. Otherwise, host names are resolved using your local HOSTS file (See “Hosts File”).

You do not need to modify the sample RESOLV file if you are not using a name server. But, any invalid entries in this file (such as an invalid IP address in the name server entry) causes name resolution to be attempted, which could result in unnecessary delays.

Note: A 4690 store controller cannot be a name server. Therefore, the RESOLV file should not identify a 4690 host Internet address. Refer to the sample RESOLV file for the format of its entries.

Hosts File

The local HOSTS file, C:\ADX_SDT1\ADXHSIHF.DAT, is searched for a match if any of these conditions exists:

- A RESOLV file does not exist or is not configured
- The name server cannot resolve the name to an Internet address
- The ADXDSN2S logical name is defined

The HOSTS file correlates a host name to an Internet address. You can have numerous host name and Internet address pairs in the HOSTS file. If you are not using a name server, you should update the HOSTS file with names and addresses of those hosts with whom this machine will normally communicate.

A sample HOSTS file is copied onto the controller when 4690 TCP/IP is installed. The HOSTS file, C:\ADX_SDT1\ADXHSIHF.DAT, can contain the following entries:

```
9.67.39.80    cc      # 4690 controller CC
9.67.39.81    dd      # 4690 controller DD
9.67.39.82    isp     # TCP/IP In-Store Processor
```

Configuring TCP Keepalive and Connection Establishment Timer Values

The TCP/IP driver enables you to configure the *tcp_keepidle* and *tcp_keepintvl* timer values. These timers are global values in the driver and are used for idle and probe interval timers for all TCP connections that have the SO_KEEPALIVE option enabled. This option can be enabled or disabled using the *setsockopt()* socket call and can be enabled by TCP applications. You can use the following logical names to define the values.

- **KEEPIDLE** - specifies the length of time to keep an idle TCP connection alive before “keepalive” probes are sent. This value is measured in minutes, with a default of 120 minutes. The valid range of values for the KEEPIDLE logical name is 1 through 120.
- **KEEPINTV** - specifies the time interval between packets sent to validate the connection (“keepalive” probes). This value is measured in seconds, with a default of 75 seconds. The valid range of values for the KEEPINTV logical name is 10 through 75.

Note: These are system-level logical names and must be defined and activated using controller configuration as User Logical File Names.

The following steps show how you could use the keepalive timer values in your socket application:

1. Assume KEEPIDLE = 5 minutes and KEEPINTV = 20 seconds.
2. In your application, use the *setsockopt()* call to enable the SO_KEEPALIVE option for your stream socket.
3. If that socket is connected to a remote host, and that connection has been idle for 5 minutes, the TCP/IP driver sends a “keepalive” probe to validate the connection.
 - a. If a response is received as a result of the probe, the connection remains active and the idle timer for that connection is reset to 5 minutes. Everything appears normal to your application.
 - b. If no response is received as a result of the probe, the TCP/IP driver continues to send probes approximately 20 seconds apart. If the TCP/IP driver does not receive a response after sending a total of eight probes, the driver closes the connection. The next time your application calls *send()* or *recv()*, an error is returned and the *tcperrno* external variable is set to ETIMEDOUT.
4. To calculate how long an idle connection remains open when the SO_KEEPALIVE option is enabled, add the idle timer value (KEEPIDLE) to the amount of time it takes to send eight “keepalive” probes, which are separated by the interval timer value (KEEPINTV).

The connection establishment timer is also configurable using logical name KEEPINIT. This timer represents the amount of time TCP waits for a new connection to be established. The default timer value is 75 seconds. To change the timer value, set logical name KEEPINIT to the new value, specified in seconds. The valid range is 10 - 75 seconds. The connection establishment timer is global and applies to all TCP connections.

Setting Up and Using Other TCP/IP Applications

This section describes how to set up and use other TCP/IP applications.

INETD Superserver (ADXHSI9L.286)

TCP/IP servers are designed to be ready to accept client connections whenever they are listening to their port number. Standard servers (for example, FTP and Telnet) listen to well-known ports awaiting incoming connections from clients.

The INETD superserver is a TCP/IP server that can listen to multiple TCP ports awaiting incoming client connections for specific servers (for example, FTP server or Telnet server). When a client attempts to connect to a specific server, INETD automatically starts the server as a 4690 background application. Using INETD enables you to have a single server continuously running instead of many servers. In addition, if using INETD to start servers, multiple instances of the same server can be concurrently active.

The C:\ADX_SDT1\ADXHSIIF.DAT file identifies which servers INETD can start. This file is copied onto the store controller during 4690 TCP/IP installation and contains an entry for the FTP server and the regular Telnet server. You must change this file if you want to use the Enhanced Telnet server, as described in "Enhanced Telnet server (ADXHSIUL.286)" on page 36.

System configuration with 4690 OS Version 4 or higher allows you to specify that the INETD superserver will be started automatically during controller IPL. If you specify automatic startup for INETD, then a background slot is not used and the INETD superserver does not have the ability to display messages on the background screen. Even if you specify automatic startup for the INETD superserver, the services that it starts continue to be started in the background.

4690 OS Version 4 or higher allows you to optionally specify a priority for each service started by the INETD server. If you adhere to the following rules, the service will start with the specified priority; otherwise, the service will start with the default priority of 5.

- Add a P=*n* string at the end of the line.
- The P=*n* string must be preceded by a blank.
- *n* must be a number from 1 to 9.

Attention: Be careful when editing the C:\ADX_SDT1\ADXHSIIF.DAT file; it must contain no blank lines. If any of the lines (including the last line) are blank, the INETD superserver abends.

You can also start the INETD superserver by configuring it as a 4690 background application, which is started when the store controller IPLs. The INETD superserver executable file name is ADX_SPGM:ADXHSI9L.286. For more information about defining 4690 background applications, see the *4690 OS: User's Guide*. Refer to the INETD data file copied onto your store controller for the format of its entries.

Note: The INETD superserver can only start TCP servers. UDP servers must be started by other means. For example, you can configure them as a 4690 background application, which starts at IPL. Therefore, if you add servers to the INETD data file, ensure that they are TCP-based servers. Also, if the server is a user-written TCP socket application and you would like INETD to start it on demand, the application should be coded to expect the socket number passed to it as the second input parameter (the first being the string "BACKGRND") in the argument list.

FTP Client (ADXHSIGL.286)

The FTP client is an interactive file transfer program invoked at the 4690 command line by issuing the command ADXHSIGL. Optionally, the "ftp" logical name is defined during the TCP/IP driver initialization to ADX_SPGM:ADXHSIGL.286 and can be used to start the FTP client.

When invoked, the FTP client displays an FTP prompt. File Transfer commands or settings can be issued at the FTP prompt. For a list of valid commands type **HELP** at the FTP prompt.

The NETRC file is an optional file that can be used by the FTP client to automate logon to certain servers and provide macro definition for frequently used command strings. The 4690 NETRC file is named

C:\ADX_SDT1\ADXHSIGF.DAT. A sample NETRC file is copied onto the store controller during 4690 TCP/IP installation. See the sample NETRC file for the format of its entries.

Note: Some FTP client operations can also be invoked with user-written applications through the use of the FTP Application Programming Interface (FTP API). See “FTP Server (ADXHSIFL.286)” for more information. See the *Transmission Control Protocol/Internet Protocol for 4690 Application Interface Guide* for more information.

FTP Server (ADXHSIFL.286)

The 4690 FTP server extends file transfer capabilities to an FTP client. Only a single FTP client can be logged on to the FTP server at any given time. Under control of the INETD superserver, a new instance of the FTP server will be started for each client logon request. This enables multiple clients to be concurrently connected to multiple servers. However, if INETD is not used and the FTP server is currently connected to a client, additional client requests will be rejected until the server ends its connection and waits for incoming connections.

The client must first log on to the server using a user ID and (optional) password. The client user ID and password are validated by the server when it finds a matching entry in the TRUSERS file. Connection requests from unauthorized clients will be rejected.

The FTP server tracks invalid login attempts per user ID. The count is reset when a successful login occurs for the user ID. If the configured number of invalid attempts is reached, the user ID is locked out for the configured duration of time.

Either an invalid user ID or an invalid password counts as an invalid login attempt. If more than 50 invalid attempts occur, all FTP users will be locked out. This condition indicates an unauthorized attack is occurring.

TRUSERS file

This file, C:\ADX_SDT1\ADXHSIU.F.DAT, should contain an entry for each client with whom the server will maintain an FTP session. You should update this file with the FTP clients to which you want to grant access.

In addition to user ID and password information in the TRUSERS file, each entry lists the read and write access privileges for that user. Access can be granted per drive or directory for physical drives A:, C:, and D: and RAM drives T:, U:, V:, and W:. In addition, the 4690 printer can be used as a file transfer destination for copying files directly from the client to a print queue, for example prn7:.

A sample TRUSERS file is copied onto the controller during 4690 TCP/IP installation. This file contains an entry for user *anonymous* (no password required) and gives read and write access to the C:\ drive and its subdirectories.

Specify multiple drives or directories by listing them on the same line, separated by a blank, as shown in the following example.

```
user: ftpuser 4690tcpip
wr: c:\anydir c:\adx_sdt1
rd: c:\
```

Note: The TRUSERS file must exist in order for the FTP server to successfully initialize. If the server cannot access the C:\ADX_SDT1\ADXHSIU.F.DAT file, it will reject the client logon request, log a system message, and then terminate.

Prior to 4690 OS Version 4, the TRUSERS file contained plain text that you could change using most text editors. To enhance security, Version 4 or higher enables you to configure FTP security using the GUI

during system configuration. You may continue to maintain the file as you did with prior versions. Once you use the GUI to configure FTP security, the TRUSERS file is converted and encrypted, and you must use the GUI to make any future changes to the file.

Quote FTP client command

An additional command has been added to the 4690 FTP server to support the starting of 4690 background applications from an FTP client. This capability is provided using the *quote* FTP client command with the ADXSTART server command. It assumes that the user at the client has the necessary execution authorization.

An example of starting a background application named TESTPGM.286 is:

```
ftp> quote adxstart testpgm.286
```

This command causes the 4690 FTP server to start testpgm.286 as a 4690 background application. The FTP server returns a message to the FTP client stating that the program has been started. The server does not wait for the background program to complete.

The program name must be fully path-qualified and is limited to 22 characters. If using path-qualified program names from a UNIX client, you might need to surround the entire quote string in double quotes if the backslash character is required. Optionally, you can use forward slash for the path name delimiter.

Program parameters can be specified immediately following the program name, each separated by at least one blank. The length of the parameter list cannot exceed 46 characters. The program will be started with default priority of 200 and cannot be changed.

User authority to start background applications using the *quote* command is granted through the TRUSERS file. A special execution access privilege is granted to clients using the ex: tag for the user. This tag is similar to the rd: and wr: tags used to grant read and write privileges. The sample TRUSERS file, ADXHSIU.F.DAT, contains an entry for user:anonymous (no password required), and grants read access (rd:) for drive C:\ (and all subdirectories of the root directory). The sample TRUSERS file also contains an entry for user:ftpuser with the password of 4690tcip, and grants write access (wr:) for drive C:\ (and all subdirectories of the root directory), read access (rd:) for drive C:\ (and all subdirectories of the root directory), and background application initiation access (ex:). This entry is shown as:

```
user: anonymous
rd: c:\
user: ftpuser 4690tcip
wr: c:\
rd: c:\
ex:
```

Note: The 'ex:' tag does not require additional parameters. Any user without execution authority will receive a message from the FTP server indicating that they are not authorized to start background applications.

Access for NFS and VFS Drives

NFS and Virtual File System (VFS) drives can also be given read and write access. To allow access for NFS and VFS drives, the drives must be listed on the same line as shown in the example below.

```
user: hayes
wr: c:\ g:\ h:\ i:\ j:\ k:\ l:\ m:\ n:\
rd: c:\ g:\ h:\ i:\ j:\ k:\ l:\ m:\ n:\
ex:
```

FTP server timeout

The 4690 FTP server contains a receive_data timeout to prevent a server hang condition in the event that either the FTP client has crashed or the physical link between the client and server is disconnected.

The timeout is set by the creation of a user-defined logical file name. To define the logical name, use 4690 controller configuration to add the name ADXFTPTO. The definition for this name can be a value between 1 and 14 400 and represents the maximum number of seconds to wait for a client to transmit data to the server.

Note: This timeout value is only relevant to the server while it is receiving data. The value does not affect either the FTP control connection IDLE time (which also can be configured from the client using the FTP site command) or have any bearing on the retransmission of data when the server is sending data to a client that might not be responding.

If the ADXFTPTO logical file name is not defined or is defined with an invalid value, a default timeout of 2 hours is used. If the timeout expires, the FTP server closes all connections with the client and terminates.

NFS server (ADXHSINL.386)

The Network File System server is a UDP-based server that enables remote access to files located in 4690 directories. An NFS client mounts an exported 4690 directory and accesses files as if they were local to the client machine. Multiple NFS clients can mount the same or different 4690 exported directories at the same time. However, only a single NFS server is required to be active to handle and keep track of all client requests.

The NFS server determines which 4690 directories it can export and which users can mount them using the EXPORTS file. You can export standard drives (A:, C:, and D:) as well as VFS drives (M: and N:). C:\ADX_SDT1\ADXHSIXF.DAT, a sample EXPORTS file, is copied onto the store controller during 4690 TCP/IP installation. You should modify this file to add, change, or delete exported directories and access privileges as well as list additional users. See “Using the EXPORTS file to give NFS clients access to files” on page 19 for examples of the format of the EXPORTS file entries. The EXPORTS file is required in order for the 4690 NFS server to initialize successfully. If the 4690 NFS server cannot obtain access to the EXPORTS file, it will log an error in the system log and terminate.

The NFS Server logs messages during start-up to controller file c:\NFSDMSG.LOG. Check this file for information if you experience problems with the NFS Server.

Notes:

1. The remote procedure call (RPC) Portmapper is a mandatory prerequisite for the NFS server. It must be active before the NFS server is started so that NFS procedures can be registered. The Portmapper is program file, C:\ADX_SPGM\ADXHSIPL.286, and can be started during a store controller IPL. Additionally, you can configure the NFS server to start during IPL. However, you must ensure that you define the Portmapper background application first, followed by the NFS server. The NFS server must be started after the Portmapper is active. Defining the Portmapper first in the configuration causes it to start before the NFS server. The NFS server program file name is C:\ADX_SPGM\ADXHSINL.386.
2. If you do not have the TZ (time zone) logical name defined and you start the NFS server from the command line, you will see a warning message indicating that the TZ environment variable is not set. This message can be ignored because the NFS server will run with or without this logical name defined. The operating system implements time zone information differently than many NFS client and server implementations. Any time and date information is passed using the *timeval* structure.
3. In order to mount the entire D: drive or A: drive, a directory named \NFSTMP must exist or be created on this drive before mounting. This does not apply to the C: drive.

Read and write block size

The NFS client decides the maximum block size for reads and writes when it mounts the remote file system (drive). Some systems might attempt a block size of 8000 bytes by default. The 4690 NFS server cannot support a block size of 8000 bytes. However, if a block size larger than 4000 bytes is required, the 4690 server will support up to 8000 bytes (not 8192) for read/write block size.

Performance

In many cases, a typical NFS server (UNIX-based) will be running on a high-performance network server machine with multiple clients concurrently mounted to multiple file systems. In addition, the high-performance server might be a dedicated file server in the network and, therefore, can have little or no other contention for its CPU. NFS clients are designed with the intention that there is a high-performance server available. It is extremely likely that a client will send multiple block-read requests simultaneously (such as if reading from the 4690 files) or transmit multiple block-write requests simultaneously (such as if writing to the 4690 files).

In addition, most clients have relatively short timeout values if a response from the server is not received. When the timeout expires, a retransmission of the block-read or block-write occurs. This timeout/retransmission sequence can actually make the load worse on the potentially already loaded server. Performance of the 4690 NFS server can be degraded under these conditions, especially considering the controller can be executing many applications of higher priority than NFS.

There are two approaches you could take if you suspect that your NFS performance could improve. First, to help alleviate potential performance degradation, you should change the NFS read/write timeout for the mounted file system (4690 drive) to a higher value than the default. For example, on the AIX® operating system, the NFS client timeout is 0.7 seconds. A recommended timeout of 7-10 seconds might help keep retransmissions low due to timer expiration. Second, consider using FTP when a large file transfer is required.

Using the EXPORTS file to give NFS clients access to files

When an NFS client attempts to access an NFS server, the server reads the EXPORTS file to determine the drives and directories that clients are allowed to access. Before NFS clients can access your NFS server, you must create an EXPORTS file using a text editor. The file name for the EXPORTS file is `adx_sdt1/adxhsixf.dat`.

Each line in the EXPORTS file lists the path to a directory. In addition, to give only certain clients access to that directory, you can list those clients following the directory. You can also specify `-ro` to limit the clients' access to read-only permission. A pound sign (`#`) at the beginning of a line indicates a comment that extends to the end of the line.

Examples:

```
# Export read/write access to everyone
c:\
m:\

# Export read-only access to everyone
d:\ -ro

# Export read/write access to hosts jack and jill
c:\ jack jill
```

Note: The previous example includes comment lines, which begin with the pound sign (`#`), to explain the intent of the line. Comment lines are optional. This example is not meant to imply that comment lines are required as part of the file.

The EXPORTS file is read only during NFS server startup. Therefore, if you change the EXPORTS file, you must stop and restart the NFS server for the changes to take effect.

NFS client (ADXHSIDL.286)

The Network File System client mounts an exported 4690 directory and accesses files as if they were local to the client machine. Multiple NFS clients can mount the same or different 4690 exported directories at the same time. The 4690 OS NFS client is needed to access the M: and N: drives from the terminal.

Configuring terminal NFS mount point data allows you to assign a terminal or group of terminals to a remote host. By assigning terminals to a mount point group and defining mount points within that group, you can define different resources to different terminals.

Within the 4690 OS V2 or higher environment, you can define a total of eight mount groups. Every terminal can be assigned to one of these mount point groups. By default, each terminal is initially assigned to mount group 8. You can define mount point data for this mount group to change the data for all terminals. Or, you can override the default by assigning various terminals to different mount groups and then defining different mount point data.

By default, no mount point data is defined for any of the mount groups. When you define mount point data for a mount group, this data is retained. For example, if you define mount point data for mount group 1 and then want to add a new terminal to mount group 1 at a later time, you do not have to redefine the fields for mount group 1, you only need to add the new terminal to the group.

Telnet client (ADXHSIVL.286)

The Telnet client support in 4690 TCP/IP enables a system to log on to a remote system using the Telnet protocol. The remote system must have a Telnet server capable of supporting VT100 type terminals. The Telnet client is a VT100 emulator and the program file name is C:\ADX_SPGM\ADXHSIVL.286.

Use the ADXHSIVL command from a command prompt to start the 4690 Telnet client. Optionally, the "vt100" (Telnet) logical name is defined during TCP/IP driver initialization to ADX_SPGM:ADXHSIVL.286 and can be used to start the Telnet client.

While in a Telnet session, the escape sequence is CTRL+]. This enables you to set or display different Telnet operating parameters. You can also exit the emulator by typing **quit** at the VT100 prompt.

BOOTP client (ADXHSIBL.286)

The BOOTP client program (C:\ADX_SPGM\ADXHSIBL.286) is used to access TCP/IP host initialization data from a BOOTP server in the network. Specifically, it can be used to obtain the 4690 IP address, domain name, name server IP address, and network router address. The BOOTP client sends a broadcast request to a BOOTP server on the network and waits for a reply. If no reply is received from a server within 25 seconds, the client gives up and does not update any network initialization data.

When a server does reply, the BOOTP client performs the following:

1. Generates a batch file named C:\ADX_SDT1\ADXPB??Z.BAT (where ?? is the local store controller/node ID) containing an *ifconfig* command that will configure the **lan0** interface for the received host IP address and, optionally, the subnet mask if one was sent. In addition, if a network router (gateway) IP address was sent by the BOOTP server, a route command is also added to this batch file.
2. Updates the resolv file with the name of the domain and IP address of the name server this host should use to resolve hostnames to Internet addresses.

Note: The BOOTP client only builds the ADXPB??Z.BAT batch file, but does **not** actually invoke the *ifconfig* and *route* commands. Prior to contacting a BOOTP server, the BOOTP client clears the IP address of the controller, so the IP configuration batch file must be run after BOOTP completes if network initialization is to occur.

To use the 4690 BOOTP client during IPL:

1. Copy the *ifconfig* *lan0* and *route* (if used) commands contained in the configuration batch file C:\ADX_SDT1\ADXIP??Z.BAT to another batch file named C:\ADX_SDT1\ADXPB??Z.BAT. (You do not need to copy the *adxhsi2l* command to ADXPB??Z.BAT file.)

2. Delete the *ifconfig lan0* and *route* (if used) commands from the ADXIP??Z.BAT file and replace them with a single command that invokes the BOOTP client (for example, adxhsibl). Next, use a call to the BOOTP batch file ADXBP??Z.

The two batch files would then look like this:

ADXIP??Z.BAT

```
adxhsi2l  
adxhsibl  
adx_sdt1:adxbp??z
```

ADXBP??Z.BAT

```
ifconfig lan0 <ip_address> ...  
route add default <router_ip_address> ...
```

Using this example, when the BOOTP server is available and does respond, the BOOTP client updates the batch file ADXBP??Z.BAT. It then returns control to the configuration batch file ADXIP??Z.BAT, which then invokes the newly created configuration.

However, in the event that the BOOTP server is unavailable or does not respond to the BOOTP client request in the timeout period, the 4690 would be able to initialize its interface based on the current contents of the ADXBP??Z.BAT file.

Note: Prior to contacting a BOOTP server, the BOOTP client clears the IP address of the controller, so the IP configuration batch file must be run if network initialization is to occur. This is the case whether the BOOTP request was successful or failed.

If your controller is configured for multiple interfaces, only the **lan0** interface can be configured using BOOTP. In this case, the *ifconfig* command for **lan1** is still included in the ADXIP??Z.BAT file as follows:

ADXIP??Z.BAT

```
adxhsi2l  
ifconfig lan1 <ip address>...  
adxhsibl  
adx_sdt1:adxbp??z
```

Using the BOOTP protocol to update/create the SNMP trap destination file

The 4690 BOOTP client can recognize a generic tag that specifies the host destination for SNMP traps. The generic tag is placed in the BOOTPTAB file on the server. It can be specified as *Tnnn*: where *nnn* is any integer between 128 and 254. The string associated with the generic tag must begin with the characters SNMP: followed by the host name or IP address of the SNMP trap destination. An example of a generic tag entry that indicates an SNMP trap destination can be found in the ADXHSAF.DAT file (BOOTPTAB) on the installation diskettes or CD-ROM.

When the 4690 BOOTP client recognizes the generic tag and determines that it is being used to specify the SNMP trap destination, it creates or updates the SNMP trap destination file, C:\ADX_SDT1\ADXHSAF.DAT, with the name or IP address of the host that follows the SNMP: prefix in the tag string. This enables automatic setting of the trap destination host during network initialization.

BOOTP server (ADXHSIAL.286)

The 4690 BOOTP server program name is C:\ADX_SPGM\ADXHSIAL.286. It services requests from BOOTP clients and supplies network initialization data, which it gathers from the BOOTPTAB file C:\ADX_SDT1\ADXHSAF.DAT. Because it is a UDP-based server, it cannot be started directly from the INETD superserver. You can, however, configure it to be a 4690 background application, which starts during the store controller IPL.

The BOOTPTAB file is a data file that contains the network initialization data for specific hosts, either based on the hardware adapter address or host IP address. A sample BOOTPTAB file is copied onto the store controller during 4690 TCP/IP installation.

The format of the BOOTPTAB file and additional details on the BOOTP server can be found in the *Transmission Control Protocol / Internet Protocol for 4690 Application Interface (TCP/IP for 4690 Application Interface)* document.

BOOTP extension file compiler

The BOOTP extension file compiler builds the extension path files described by RFC1497.

Configuring the PXE Environment and DHCP Server (DHCPD.386)

The Preboot Execution Environment (PXE) provides a consistent and industry-standard means for loading terminals. PXE provides an additional protocol for loading terminals versus the proprietary RPL protocol currently used for loading terminals. PXE uses a standard set of TCP/IP protocols, namely Dynamic Host Configuration Protocol (DHCP) and Trivial File Transfer Protocol (TFTP). DHCP and TFTP provide network address allocation and network-based booting on the client terminals.

PXE Environment Restrictions

The following restrictions apply to the PXE environment:

- PXE booting is supported only on **Ethernet-attached** terminals.
- PXE booting is **required** on the 4694-2x7, 4694-3x7 terminal models, SurePOS models 350, 72x, 74x, 77x and 78x, as well as TCxWave 6140 Series terminal models, if they are Ethernet attached.
- In addition, PXE booting is supported on the 4694-205, 4694-206, 4694-245, 4694-246 terminal models, and on all SurePOS 300/700 series terminal models.
- Because PXE/DHCP involves IP broadcasting, the network should be configured in such a way as to prevent "store hopping," that is allowing DHCP requests to be fulfilled by controllers in other stores. In a non-routed network, "store hopping" can be prevented by configuring each store on its own IP subnet.
- On a multihomed store controller, PXE booting is supported on the TCC (or **lan1**) interface only.
- The 4690 DHCP Server only responds to DHCP requests from the terminals listed above as being supported in a PXE environment. It is not intended for use as a general purpose DHCP server.

PXE/DHCP/TFTP Server Communication

The flow for booting the 4690 terminals and how PXE, DHCP, and TFTP exchange information is described below.

- The PXE/DHCP client running on the 4690 terminals uses DHCP to acquire an IP address and boot information.
- Upon receiving configuration and boot information from the DHCP exchange, the PXE client uses TFTP or Multicast TFTP (MTFTP) to download the bootstrap. A TFTP server must be running on the controller.

Note: Multicast TFTP protocol is based on the standard TFTP protocol but, in this case, the requested file is multicast to the client or to multiple clients who have joined the multicast group.

- When the bootstrap code gains control, it uses MTFTP to download the operating system.

The flow for dumping the 4690 terminals and how PXE, DHCP, and TFTP exchange information is described below.

- The PXE client running on the 4690 terminal uses DHCP to acquire an IP address.
- The PXE client uses TFTP to send the dump to the controller that is assigned the IP address.

Note: The primary controller that is configured for a PXE terminal is not necessarily the controller that the terminal dumps to. When the terminal wants to dump, a request for an IP address is broadcast. All DHCP servers on the store's subnet respond. The first controller that the terminal receives a response from is typically the controller that the terminal dumps to.

Note: The dump is transferred in a compressed format from the terminal to a temporary file on the controller named ADX_SDT1:ADXDM???.DAT, where ??? is the terminal number. On the controller, the dump is automatically uncompressed by a process called the PXE Dump Decompressor, ADXPXEDL.286. The dump is uncompressed to a file named ADX_SDT1:ADXCSTF.DAT, which can then be used for problem analysis.

PXE Load Performance Considerations

In order to optimize load times for PXE terminals, the DHCP Server is configured to **not** check for in-use addresses.

If the need to protect against duplicate IP addresses on the network is more critical, the DHCP Server can be configured to check for in-use addresses before assigning an IP address to a 4690 terminal. This configuration is done by including the pingTime parameter with a value greater than zero in the DHCP Server configuration file. The pingTime parameter with a nonzero value tells the DHCP server to ping an IP address to insure that it is not already in use before offering the IP address. The pingTime value specifies the amount of time (in milliseconds) that the DHCP Server waits for a response after pinging the IP address. Because the in-use check causes a delay in servicing DHCP requests, the recommendation is to specify a value of 200 ms or less. The DHCP Server configuration file is named ADXIPxxD.DAT, where xx is the node ID of the controller. The syntax to configure this item is: pingTime 200.

Note: The pingTime parameter is not configurable using the DHCP Server Configuration panels in Controller Configuration.

PXE Configuration

Complete the following steps to configure the PXE environment:

1. Change the boot protocol in your terminal setup to PXE. Because this process is different for each terminal type, refer to the manuals that shipped with your terminal for instructions on how to change the boot protocol.

Note: If the terminal has a hard disk drive, disable the hard disk drive in the boot sequence in your terminal setup.

2. Ensure the 4690 controller is configured for TCP/IP. See “Configuring 4690 TCP/IP” on page 10 for additional information.
3. Configure the DHCP server environment on the 4690 controller as described in “Configuring the DHCP Server.”

Configuring the DHCP Server

The DHCP server can be configured by using either the GUI configuration panels under Controller Configuration or by editing a text configuration file using a text editor. The configuration data is for the operation of the DHCP, TFTP, and the PXE Dump servers on the controller.

To use the DHCP Server GUI, choose **Controller Configuration** from the main menu and then choose **DHCP Server**. In this case, the correct file name is created or modified for the controller that is being configured. The DHCP Server GUI saves the changes to an inactive file. These changes only become active after activating the controller configuration.

The DHCP server can also be configured by editing the active DHCP server configuration file. To manually edit the DHCP configuration file, a sample DHCP configuration file is provided. This sample configuration file specifies the necessary parameters for PXE booting and IP address allocation. This file can be modified easily using any text editor. The sample DHCP configuration file is named ADX_SDT1:ADXIP00D.DAT and is shown in Figure 1 on page 24.

To begin, copy the sample configuration file(ADX_SDT1:ADXIP00D.DAT) to the active DHCP configuration file, ADX_SDT1:ADXIPxxD.DAT, where xx is the 2-character controller node ID. For example, if your controller node ID is CC, the copied file name will be ADX_SDT1:ADXIPCCD.DAT.

After modifying the DHCP server configuration (using either the DHCP Server GUI followed by controller configuration activation or by editing the active file), the configuration can be applied by either IPLing the controller or by using the update option of the DHCPINFO Utility (dhcpinfo -u). For additional details about using this feature, see “DHCPINFO utility” on page 27.

Note: The function of the StartPXEServers parameter is to start the PXE servers during controller IPL. If you change the value of the parameter, it only has an effect when the controller is IPLed.

Notes:

1. The active DHCP server configuration file name is ADX_SDT1:ADXIPxxD.DAT and is used when editing the DHCP server configuration file. The inactive file name is ADX_SDT1:ADXIPxxI.DAT, which is created by the DHCP Server GUI and is copied to the active file name upon controller configuration activation. In both file names, xx is the 2-character controller node ID.
2. See the *4690 OS: Planning, Installation, and Configuration Guide* for more information on how to use the DHCP Server GUI.
3. Because the DHCP Server GUI has error detection capability, any errors made when manually editing the DHCP server active configuration file (ADX_SDT1/ADXIPxxD.DAT) could prevent the DHCP Server GUI from starting. If this situation occurs, delete the inactive configuration file (ADX_SDT1/ADXIPxxI.DAT) and manually correct the error in the active configuration file.
4. If you manually create the DHCP configuration file, the sample comments are also viewable when you use the GUI.

```
# This file is a sample configuration file for a DHCP server supporting
# PXE clients. Comments are added to assist in understanding the
# configuration file.
# Start the servers required for PXE booting automatically: yes or no
StartPXEServers yes
# PXE boot file name
option 67 c:/adx_spgm/adxpxebl.btt # boot file name
option 13 64 # boot file size
# Global options. Passed to every client unless overridden at a lower scope.
# option 15 "test.ibm.com" # domain name
# option 6 10.1.1.1 # dns server
# option 3 10.1.1.1 # default router
# Setup to send options to the pxeclient. Sent in the encapsulated option 43.
vendor PXEClient
{
  option 1 224.46.1.1 # multicast address for PXE bootstrap
  option 2 76 # mtftp client - port 76
  option 3 75 # mtftp server - port 75
  option 4 2 # open timeout
  option 5 2 # reopen timeout
  option 200 224.46.1.2 # multicast address for OS load
}
# The following defines the IP address pool that the DHCP Server uses
# to assign IP addresses to DHCP clients. The format is as follows:
# subnet <subnet address> <netmask> <address range>
subnet 10.1.1.0 255.255.255.0 10.1.1.3-10.1.1.30
{
  option 1 255.255.255.0 # netmask
}
#
# end of adxip00d.dat
#
```

Figure 1. Sample DHCP Configuration File

Quick DHCP Setup for PXE Booting

To setup DHCP quickly for PXE booting, complete the following steps:

1. Set the StartPXEServers parameter to yes in the controller's DHCP configuration file. This parameter automatically starts the DHCP, TFTP, and PXE dump servers during controller IPL. Setting this parameter to yes is required for terminal loading.

Note: Ensure that the TFTP server (ADXHSITL.286) is not configured as a controller background application. If the TFTP server was configured previously as a controller background application, it must be unconfigured.

2. Specify the IP subnet and address range for the IP address assignment by changing the values after the subnet keyword in the controller's DHCP server configuration file. See "Configuring values for subnet keyword" on page 26 for additional details.
3. Re-IPL the controller and you are ready for booting terminals using PXE.

DHCP Server Backup Configuration

To have backup capability for loading and dumping PXE terminals and/or for DHCP-assigned IP addresses, and in addition to the prior configuration instructions, you can configure a DHCP server to be running on more than one controller at the same time.

For each controller that will have DHCP running :

- Generate a DHCP configuration file (ADX_SDT1/ADXIPxxD.DAT, where xx is the controller node ID).
- Set StartPXEServers to yes in each DHCP server configuration file.
- The multicast addresses (PXEClient vendor options 1 and 200) must be unique for each DHCP server configuration file.
- The address range (on the subnet line) must be unique for each DHCP server configuration file.
- Re-IPL the controller.

DHCP Configuration Options

The following list represents the set of options available for DHCP configuration in the DHCP server configuration file.

- StartPXEServers - yes or no; indicates whether to automatically start the DHCP, TFTP, and PXE dump servers during controller IPL
- subnet - IP subnet/address range (see "Configuring values for subnet keyword" on page 26)
- vendor - vendor class (only "PXEClient" vendor is supported)
 - PXEClient vendor option 1 - multicast IP address for loading bootstrap
 - PXEClient vendor option 2 - multicast client UDP port
 - PXEClient vendor option 3 - multicast server UDP port
 - PXEClient vendor option 4 - multicast open timeout
 - PXEClient vendor option 5 - multicast reopen timeout
 - PXEClient vendor option 200 - multicast IP address for loading 4690 OS
- option 1 - network mask (not required for PXE environment)
- option 3 - router IP address (not required for PXE environment)
- option 6 - name server IP address (not required for PXE environment)
- option 13 - boot file size (always set to **64**)
- option 15 - domain name (not required for PXE environment)
- option 67 - boot file name (always set to **C:/ADX_SPGM/ADXPXE.BTT**)

Note: The PXEClient vendor options should be changed under the following conditions:

- The multicast IP addresses/ports (PXEClient vendor options 1, 2, 3, and 200) conflict with other multicast IP addresses/IP ports on your LAN. For operation in a wireless environment, the multicast IP addresses must use 224.46.xxx.xxx.
- You are running multiple, concurrent 4690 DHCP servers on the same LAN, in which case the multicast IP addresses (PXEClient vendor options 1 and 200) must all be unique. These addresses can be changed by simply changing the last field of the IP address.

Note: One or more subnet statements are allowed in a configuration file.

Configuring values for subnet keyword

subnet <Subnet address> [<Subnet Mask>] [<range>]

The Subnet address is the address of the subnet of this controller as configured in ADXIPxxZ.BAT. The subnet address is specified in dotted decimal notation, for example, 9.17.32.0 or 128.81.22.0. The subnet must be within the subnet mask, and the address can be no longer in bits than the subnet mask. For example, if the subnet mask is 255.255.255.0, the address 9.67.10.128 is too long. The subnet address can be optionally followed by the Subnet Mask or a range.

The Subnet Mask for the subnet is specified in dotted decimal notation or in integer format. A subnet mask divides the subnet address into a subnet portion and a host portion. If no value is entered for the subnet mask, the default is the class mask that is appropriate for an A, B, or C class network. The subnet mask is specified in dotted decimal notation, for example, 255.255.255.0.

If a range is specified, it describes the pool of addresses that this server will administer for this subnet. A range is specified by host addresses in dotted decimal notation separated by a hyphen with no intervening spaces, for example, 192.81.20.1-192.81.20.128. If no range is specified, all host addresses in the subnet are administered by this server.

Notes:

1. The address pools administered by different DHCP servers on the same subnet, that are running at the same time, must not overlap; otherwise, two hosts could be assigned the same address.\
2. Statically assigned IP addresses that are defined in the terminal load definition must not overlap with the DHCP server address pool.

For example, to configure the range of addresses 192.81.20.1 through 192.81.20.127 on subnet 192.81.20.0 with subnet mask 255.255.255.0, create the subnet statement:

```
subnet 192.81.20.0 255.255.255.0 192.81.20.1-192.81.20.127
```

A subnet statement can be followed immediately by a pair of curly brackets, in which parameters particular to this subnet can be specified, such as options.

Configuring wireless terminals for PXE

For additional information, see the *4690 Wireless Tips* in the Knowledgebase on the Toshiba support site.

Configuring DHCP for terminal TCP/IP

When the terminal has loaded the operating system, DHCP can be used to configure operational IP for use by TCP/IP applications running on the terminal. In this case, a DHCP client runs on the terminal during terminal IPL to request an IP address and other IP configuration information. To co-exist with multiple DHCP servers on the network, the DHCP client can be tailored to limit the DHCP negotiation to only a 4690 DHCP server or to negotiate with non-4690 DHCP servers. By default, the DHCP client accepts **only** an offer from a 4690 DHCP server.

To configure the DHCP client to accept offers from non-4690 DHCP servers, a terminal logical name must be defined and named DHCPEXT in ADX_IDT1:ADXTRMUF.DAT. DHCPEXT must be set to one of the following values:

- DHCPEXT = 1 - Client is serviced by the first offer that is received from any DHCP server
- DHCPEXT = 2 - Client is serviced by only non-4690 DHCP servers

DHCP for TCP/IP terminal configuration is available for Ethernet-attached terminals. First, in the terminal load definition under the TCP/IP option, indicate 1 (yes). Second, under the Address Method option, indicate 2 (DHCP server). In this case, the following options are available in the DHCP server configuration file to configure your terminal for TCP/IP. These are the same fields that would be configured under static TCP/IP configuration.

- option 1 - subnet mask
- option 3 - default router IP address

- option 6 - name server IP address
- option 15 - domain name

If a terminal is configured for TCP/IP using DHCP, the HOSTS file, ADX_SDT1:ADXHSIHF.DAT, is updated by the DHCP server with the assigned hostname and IP address. The terminal hostname has the format Tssssttt where ssss is the store number and ttt is the terminal number. For example, terminal 005 in store 0055 is assigned IP address 10.10.1.4, and would add the following entry to the HOSTS file:

```
10.10.1.4 T0055005 #DHCP
```

This entry allows TCP/IP applications to address the terminal as host name, T0055005, rather than the dotted decimal IP address. The #DHCP comment indicates that this entry was added by the DHCP server and therefore, can be "managed" by the DHCP server. The user can still add alias hostnames of their own choosing to the HOSTS file without the #DHCP comment. These alias hostnames in the HOSTS file are not managed by the DHCP server.

DHCPINFO utility

DHCPINFO is a utility for displaying DHCP Server configuration information and lease status, for removing leases, and for applying updates to the DHCP server configuration. The format for the DHCPINFO Utility is shown below.

```
dhcpinfo [-s <server IP address>] [-r <IP address> | "all"] [-u]
          [-t <RTU server IP address> | "0"] [-b "supps" [-v <minutes>] | -b "0"]
```

Where:

dhcpinfo with no options displays DHCP server configuration information and lease status on the local machine.

-s specifies a remote server IP address for displaying status, removing leases, updating the DHCP server, RTU configuration or supplemental boot mode configuration.

-r specifies the IP address of an individual lease to remove or "all" for removing all leases.

-u updates the DHCP server configuration using the current active configuration file. This option allows you to apply a modified DHCP configuration without a controller IPL.

-t applies the DHCP Remote Terminal Utility (RTU) option with the specified RTU Server IP address or removes the applied RTU option if 0 is specified. The DHCP RTU option is active only in DHCP server memory and is not retained on a controller IPL. Also, dhcpinfo -t internally performs a DHCP configuration update (dhcpinfo -u) to apply the RTU option settings; therefore any modifications in the active DHCP Server configuration file will also be applied with dhcpinfo -t. Refer to "Using the Remote Terminal Utility" on page 31.

-b **supps** applies the DHCP option to enable network supplemental boot or removes the option if 0 is specified. By default, supplemental boot mode is enabled for a period of five minutes and is automatically disabled after this time period. For more information on the network supplemental boot option, refer to "Booting Enhanced Supplementals over the network" in the 4690 OS: User's Guide.

-v **-b supps** parameter to set a time-out period (0-60 minutes) for the supplemental boot mode other than the default of 5 minutes. A value of 0 indicates no time-out on supplemental boot mode and the user may manually disable using the **-b 0** parameter.

Examples:

dhcpinfo

displays the status of the DHCP server on the local machine

dhcpinfo -s 1.1.1.2

displays the status of the DHCP server that is running at IP address 1.1.1.2

dhcpinfo -r 1.1.1.5

removes the lease for the IP address 1.1.1.5 (local server)

dhcpinfo -r all

removes all the leases assigned by the local DHCP server

dhcpcinfo -r 1.1.1.10 -s 1.1.1.2
removes the lease for IP address 1.1.1.10 that was assigned by server 1.1.1.2

dhcpcinfo -u
updates the configuration of the DHCP server running on the local machine

dhcpcinfo -u -s 1.1.1.2
updates the configuration of the DHCP server running at IP address 1.1.1.2

dhcpcinfo -t 1.1.1.1
sets DHCP RTU option to 1.1.1.1 on local DHCP server

dhcpcinfo -t 0
clears DHCP RTU option on local DHCP server

dhcpcinfo -t 1.1.1.1 -s 1.1.1.2
sets DHCP RTU option to 1.1.1.1 on DHCP server 1.1.1.2

dhcpcinfo -t 0 -s 1.1.1.2
clears DHCP RTU option on DHCP server 1.1.1.2

dhcpcinfo -b supps
enables network supplemental boot on the DHCP server on the local machine for a period of 5 minutes

dhcpcinfo -b supps -v 10
extends the period of time the supplemental boot is enabled to 10 minutes

dhcpcinfo -b supps -v 0
enables network supplemental without a timeout period

dhcpcinfo -b supps -s 1.1.1.2
enables network supplemental boot on DHCP server that is running at IP address 1.1.1.2

dhcpcinfo -b 0
disables network supplemental boot on the DHCP server on the local machine

dhcpcinfo -b 0 -s 1.1.1.2
disables network supplemental boot on the DHCP server that is running at IP address 1.1.1.2

Following is a sample of output from the DHCPINFO Utility along with descriptions of the output fields.

DHCP Status for Server 1.1.1.2:

RTU Server: 1.1.1.2

Boot Mode: supps

Address	Status	Lease	Client ID	Last Leased
-----	-----	-----	-----	-----
1.1.1.5	In Use	1000	0004acd12345	
1.1.1.6	Leased	86400	0004acd23456	Oct 3 16:24:57 2001
1.1.1.7	Reserved	300	0004acd34567	Oct 3 16:25:01 2001
1.1.1.8	Leased	Infinite	0004acd45678	Oct 3 16:27:59 2001
1.1.1.9	Released			
1.1.1.10	Available			

Address:

The IP addresses that are configured for assignment by this DHCP server.

Status:

- Available - The address is available for assignment by the DHCP server.
- Leased - The DHCP server has leased the address to a DHCP client.
- Released - The DHCP client has released the address.
- Reserved - The DHCP server has issued an offer to a DHCP client for this address.
- Expired - The lease has expired.
- In Use - The DHCP server has found this address to be in use.

Lease: The amount of time in seconds of the lease for this address. The amount of time might also be "Infinite" if it is a permanent lease. Reserved and in use addresses are also timed with a short lease.

Client ID:
The MAC address of the client.

Last Leased:
The time when the DHCP server last assigned the address.

RTU Server:
The IP address that is configured for RTU by this DHCP server or none if an RTU server is not configured.

Boot Mode: supps
The network supplemental boot option is enabled on this DHCP server. This line will be blank if not enabled.

Troubleshooting the DHCP server

The DHCP server logs messages W978 and W980 to aid in problem resolution. Event number = E200 indicates that the DHCP server logged the message. For the event numbers, see Table 7.

Table 7. DHCP server error message event numbers

Call	Info	RC	Description
02xx	DHCP INIT		The DHCP server could not initialize due to TCP/IP socket errors. Ensure that TCP/IP is configured and running on your controller. This message could also occur if you attempt to start the DHCP server when it is already running.
0301	<i>config file name</i>		The DHCP server could not open the DHCP server configuration file.
0402	DHCP NO IP		The DHCP server has no IP addresses available for assignment. All IP addresses in the configured address pool have been assigned by the DHCP server or are in use, such as assigned by another DHCP server or statically assigned. Use the DHCPINFO Utility to determine the status of the DHCP IP address assignment and whether the IP address pool must be increased to cover the number of terminals. This error can also indicate that the subnets configured in the DHCP Server Configuration do not include the subnet of the host (terminal) requesting an IP address. If assigning addresses on a local subnet only, the local subnet is determined by the IP address of the controller where it is running as configured in ADXIPxxZ.BAT (IP address of TCC interface or lan1 , if multihomed). In this case, review the ADXIPxxD.DAT and ADXIPxxZ.BAT files and ensure that the controller IP address and the DHCP address range to be assigned to terminals are all on the same subnet.
0403 / 0404	DHCP INIT		The DHCP server could not initialize due to errors in the DHCP server configuration. Review the ADXIPxxD.DAT file (where xx is the controller node ID) for possible errors. A common problem is that the address range specified on the subnet statement is outside the scope of the subnet.
0405	<i>hostname</i>	<i>IP address in hexadecimal format</i>	The DHCP server could not update the HOSTS file with the hostname and IP address. Most likely this is a result of no acting master.

Using TCC and PXE in a routed environment

If communications between the 4690 store controller and the terminal pass through a routed network, the 4690 environment must be configured to use IP protocols. First, TCP/IP must be configured on the controller and terminals, including configuration of the default router. In addition, the terminals must be configured for TCC over IP (TCC/IP), and the terminals must be booted using PXE. Additional changes to configure the TCC/IP and PXE protocols for routing are described below.

Because TCC over IP (TCC/IP) uses IP multicasting for communications between the controller and the terminal, the IP Multicast TTL (time-to-live) must be adjusted for use over a router. TTL is the number of hops or routers through which a packet passes before being discarded. By default, the Multicast TTL is set to 1 to avoid unwanted forwarding of multicast packets. When using TCC/IP over a router, the TCC Multicast TTL must be increased to the number of hops on the TCC network. The TTL should be set to the minimum value required to pass through any in-store routers to prevent multicast traffic from being forwarded to other stores. In addition, a TCC Acknowledgment Time-Out value can be adjusted (although, this adjustment is not required) to account for any delays introduced on the TCC network due to the presence of routers. These values are set by editing the TCC/IP configuration file, `ADX_SPGM:ADXTCCIF.DAT`, shown below. Details for modifying the TCC/IP configuration are described in the TCC/IP configuration file itself.

PXE booting uses DHCP to provide IP configuration information to the PXE client on the terminal. If passing through a router, the Router option, DHCP option 3, in the DHCP Server Configuration should be configured with the IP address of the router. PXE booting uses Multicast TFTP (MTFTP) for loading terminals, and uses the same TTL value configured in the TCC/IP configuration file to set its Multicast TTL.

Configuration changes on the router are also required. The router must be configured to forward the DHCP requests to the DHCP server on the store controllers. Depending on the actual router, these configuration changes would include configuring the "DHCP (or BOOTP) Relay Agent" and the "Helper Addresses." Also, the router must be configured for multicast support, including enabling multicast and specifying that IGMP Version 1 is to be used.

TCC over IP configuration file (`adx_spgm:adxtccif.dat`)

```
# This file contains various default values for TCC over IP
# and is used by both controller and terminal.
# Comment lines begin with '#', edited values MUST begin in
# the leftmost column. You may edit any of the default values
# but DO NOT DELETE ANY OF THE LINES CONTAINING A DEFAULT
# VALUE OR CHANGE THEIR ORDER WITHIN THIS FILE. Any value
# which does not fall within the specified ranges will be
# assigned the default value.
# To activate changes to this file, build a new terminal
# load shrink, re-ip controllers and reload terminals.
#
# The following is an explanation of the default
# values which are related to TCP/IP:
#
# IP Multicast addresses must fall within the range of
# 224.0.0.0 - 224.255.255.255 however do not use
# well known IP multicast addresses.
# Port numbers must fall within the range of 1024 - 65535
#
# CONTROLLER MULTICAST ADDRESS is the multicast group
# address which the controller joins.
# TERMINAL MULTICAST ADDRESS is the multicast group
# address which the terminal joins.
# PORT numbers are used by both controller and
# terminal for multicast and acknowledgments.
# These port numbers should not be configured in the ports file,
# ADXHSISF.DAT.
# IP MULTICAST TIME TO LIVE is the number of hops which a
# multicast can travel before it is discarded from the network.
```

```
#
# controller multicast address
224.46.90.1
# terminal multicast address
224.46.90.0
# multicast port
4691
# acknowledgment port
4692
# IP multicast time-to-live, range 1-255
1
#
# The following is an explanation of the default value which
# is related to TCC:
#
# ACK TIME OUT is used by a TCC network station to detect a failure
# to receive a required acknowledgment from another TCC network station.
# The duration of the timer should take into account the maximum delay
# between a controller and terminal in your TCC network. Also take into
# account that increasing the ack time out will increase, by a factor of 3,
# the time for a terminal to go offline and the time for the backup to
# respond. Normally, the default value of 1000ms (1 sec) is more than
# adequate for this delay, however, if a TCC network station requires a
# delay greater than 1 second, the value can be edited. The timer
# has a granularity of 32ms.
#
# ack time out, range 1000 - 99999
1000
```

TFTP and MTFTP servers

TFTP is a simple file transfer protocol. The TFTP server services requests from TFTP clients by transferring files to or from the clients. TFTP is mainly used for bootstrapping medialess clients. FTP should continue to be used for normal file transfer purposes. In addition, the 4690 TFTP server supports multicast file transfer. Additional details can be found in the *TCP / IP 4690 Application Interface* document.

Because the TFTP server is a UDP-based server, it cannot be directly started from the INETD superserver. To configure TFTP for PXE terminal loading, see “TFTP and MTFTP server startup and configuration for PXE booting” for configuration information. Otherwise, you can configure TFTP to be a 4690 background application that starts during the store controller IPL.

TFTP and MTFTP server startup and configuration for PXE booting

The MTFTP server uses the DHCP configuration file to determine both the multicast address and the transfer file name. The TFTP and MTFTP servers are required for PXE booting and are started automatically when the StartPXEServers option is set to yes in the DHCP configuration file.

Starting with Version 5, the TFTP server is updated to address security concerns with unrestricted file access. When automatically started for PXE, the TFTP server restricts TFTP file access to only the ADX_SPGM and ADX_BOOT directories and PXE dump files. Also, it does not allow overwrite of existing files.

The MTFTP server requires two multicast addresses for PXE support; one to multicast the bootstrap and a second one to multicast the 4690 OS load. The MTFTP server reads this information from the DHCP configuration file. Refer to “Configuring the PXE Environment and DHCP Server (DHCPD.386)” on page 22 for information regarding starting and configuring TFTP and MTFTP servers for PXE booting.

Using the Remote Terminal Utility

The Remote Terminal Utility (RTU) provides the means to remotely load and run a utility on a PXE terminal prior to loading the OS. This capability is useful for remotely configuring CMOS settings on unattended terminals. RTU is supported on SurePOS Model 350, Model 72x, 74x, 77x and 78x terminals and on TCxWave 6140 Series models. RTU can be configured to run on one or more terminals and can

run the same or different utilities. The configured utility runs one time only on the next terminal reload and subsequently loads the OS. Further terminal reloads will not run the utility unless it is reconfigured. Details for configuring RTU are provided below. Information for creating an RTU compatible utility image is provided in “Creating the RTU image” on page 33.

Note: Please take precautions when using any utility. RTU cannot safeguard against improper usage of a utility, using the wrong utility, or a utility with incorrect data. Such cases of improper usage could result in terminals not being able to load and might require service personnel to resolve.

The RTU server (adxrtu.386) is a 32-bit application which runs on the store controller and communicates with the PXE bootstrap to provide RTU information. The RTU server may be started from the command line or as a background application.

The user provides a configuration file as input to the RTU server to indicate the list of terminals and utilities to be processed. The RTU configuration file, `adx_sdt1:adxrtucf.dat`, is a local text file created on the controller running the RTU server. The file contains a variable number of lines, each line with a terminal number and name of the utility image file to be loaded. White space is ignored and '#' in the first position of a line indicates a comment line. Example RTU configuration entries are shown below to load `cmos721.img` on terminal 100 and `cmos722.img` on terminal 105.

```
100 adx_boot:cmos721.img
105 adx_boot:cmos722.img
```

A DHCP option is used to trigger RTU requests on the terminal. The DHCP RTU option includes the IP address of the controller running the RTU server. The user configures the RTU DHCP option on each controller running a DHCP server. When a PXE terminal is booted, the RTU DHCP option is included with the DHCP information provided to the terminal PXE bootstrap and indicates to request RTU information from the RTU server running at the IP address specified.

The DHCP RTU option is configured in one of two ways. The first method is to update the DHCP Server Configuration file with the `rtuserver` keyword followed by the IP address of the controller running the RTU server as shown below:

```
rtuserver 1.1.1.1
```

When the DHCP configuration file has been modified, the DHCP configuration is applied using either `dhcpinfo -u` or by a controller IPL.

Another method of configuring the DHCP option is to use the `-t` parameter of the `dhcpinfo` command to enable or disable RTU dynamically. Refer to “DHCPINFO utility” on page 27 for more information.

If the RTU option is enabled, the terminal sends a request to the RTU server to inquire if it should load a special image. The server responds with either an image name to load or indicates to continue booting normally. The terminal uses TFTP to transfer the image and notifies the server of the success or failure of the load. If successful, the terminal passes control to the RTU image and the server updates its list of terminals to be processed indicating this terminal has completed its RTU image load. The server also modifies the RTU configuration file with a comment character (#) in column 1 of that entry. If the RTU image does not load, the terminal remains in the RTU list to be attempted again on the next terminal reload and the terminal continues to boot normally.

The RTU server maintains a log with time-stamped entries indicating success or failure of terminals loading RTU images. Other RTU status, errors and informational messages are also logged by the RTU server to maintain a history of RTU activity.

Note: The results of the actual utility started by the RTU service are not available to 4690 OS.

Instructions for running RTU

The list below summarizes the steps required to configure and run the RTU service:

1. Create the image file (such as CMOS set utility) and install on the controller in the c:/adx_boot directory and distribute to other controllers.
2. Create the configuration file, adx_sdt1:adxrtucf.dat, with terminal number(s) and image name(s) to load.
3. Start the RTU server, adxrtu.386, from either command line or background.
4. Configure the DHCP RTU option on each DHCP server using dhcpinfo -t x.x.x.x [-s y.y.y.y] where
 - x.x.x.x is the dotted decimal IP address of the RTU server.
 - y.y.y.y is the IP address of the DHCP server (-s option not required if the DHCP server is the local controller).
5. Load terminals. Terminals will run the image, reboot and load normally.
6. Check the output file, adx_sdt1:adxrtu.log, to determine status.
7. Issue dhcpinfo -t 0 to all DHCP servers to disable RTU checking on the terminal side. Stop the RTU server.

Creating the RTU image

The RTU image is created as follows:

1. Create a DOS bootable diskette containing the following:
 - the DOS utility to run on the terminal (for example, the CMOS set utility).
 - coldrbt.com* to reload the terminal after the utility has run (optional).
 - autoexec.bat file invoking the utility and optionally invoking coldrbt.com.
2. On a Windows system, create the RTU image from the DOS diskette created above using the savedskf* utility as follows:
 - savedskf a: <filename> /a/d

Note: * coldrbt.com and savedskf.exe are available on the 4690 OS Optional Programs package.

SNMP agent (ADXHSI1L.286)

The 4690 Simple Network Management Protocol (SNMP) agent (server) is designed to operate with an SNMP network management station (client). The SNMP network management station polls or queries the SNMP agent for information about the network (for example, number of UDP datagrams transmitted). The information is stored by the agent in a database called the Management Information Base, or MIB. The 4690 SNMP agent supports the MIB-II definition for managed objects (excluding the Exterior Gateway Protocol group) for GET and GET-NEXT requests. The setting (SET-request) of MIB objects is not supported.

In addition to providing access to MIB-II network objects, the SNMP agent can asynchronously send TRAPs to the network management station whenever a significant network event occurs. Currently, there are two different types of TRAPs that can be sent by the SNMP agent:

- Cold-Start, which notifies the network management station that this SNMP agent has just been started
- Authentication Failure, which indicates that an attempt to access this SNMP agent's MIB objects was made by an unauthorized network management station

Because the SNMP agent is a UDP-based server, it cannot be started by the INETD superserver. However, it can be configured as a 4690 background application, which starts during the store controller IPL. See the *4690 OS: User's Guide* for information on configuring background applications.

Community names file

Access to the 4690 MIB-II objects is only allowed for authorized network management stations. Those stations are defined in the community names file.

The 4690 SNMP agent uses an encrypted community names file, C:\ADX_SDT1\ADXHSIEF.DAT, to validate a network management station. The encrypted file is built from an ASCII text file, C:\ADX_SDT1\ADXHSIQF.DAT, created by the user.

To build the encrypted community names file:

1. Create the ASCII text community names file as C:\ADX_SDT1\ADXHSIQF.DAT. (A sample file is copied onto the store controller during 4690 TCP/IP installation. Refer to this sample file for the format of its entries.)
2. Invoke the encryption program ADXHSI8L from a 4690 command line. This program assumes the input file is C:\ADX_SDT1\ADXHSIQF.DAT and creates the encrypted output file C:\ADX_SDT1\ADXHSIEF.DAT.

It is recommended that you build and maintain the ASCII text community names file in a secure location. Only the encrypted community names file needs to exist on the 4690 store controller.

Note: The encrypted community names file must exist in order for the 4690 SNMP agent to successfully initialize. If the SNMP agent cannot access the encrypted community names file, it will log a system message and terminate.

SNMP agent environment names

In addition to the community names file, three environment names must be defined to the 4690 system. These names correspond directly to MIB objects. The environment names are:

- **syscont** - specifies who to contact in case of problems, for example, Joe Smith.
- **sysloc** - specifies the physical location of this host, for example, Store 101CC.
- **hostname** - specifies the name of the host on which you are running. This name must be able to be resolved to an IP address either by a name server or using the hosts file.

These three environment names are defined by one of two methods: user-defined logical file names in 4690 controller configuration or listed in the C:\ADX_SDT1\ADXHSIZF.DAT file.

The format of the SNMP environment names file, C:\ADX_SDT1\ADXHSIZF.DAT, is:

```
syscont  Contact_Name    # comment
hostname  Host_Name      # comment
sysloc    System_Location # comment
```

A sample SNMP environment names file, C:\ADX_SDT1\ADXHSIZF.DAT, is copied onto the 4690 store controller during 4690 TCP/IP installation. Precedence is given to the logical file name definition of the SNMP environment name before determining if the name is defined in the SNMP environment names file.

Note: The three SNMP environment names must exist on each machine in which the 4690 SNMP agent is to run either as user-defined logical file names or within the SNMP environment names file (or some combination of both). However, the assignment for each name does not have to be unique across all machines (for example, a syscont of Joe Smith might not be the same on every machine).

SNMP trap destination file

The SNMP trap destination file identifies the network management station to which traps will be sent. This file is named C:\ADX_SDT1\ADXHSIDF.DAT. A sample trap destination file is copied onto the store controller during 4690 TCP/IP installation. Refer to the sample trap destination file for the format of its entries. If the trap destination file does not exist, traps will not be sent.

MIB-II

Extension of the MIB objects can be accomplished using the SNMP Distributed Program Interface (SNMP DPI) for 4690. The SNMP DPI enables enterprise-specific MIB objects to be created and also provides an application the capability to generate traps. Refer to your TCP/IP documentation for information on MIB-II groups and objects.

Telnet server

The 4690 Telnet server can support a VT100, ANSITERM, or 3151 Telnet client. The following 4690 Telnet server characteristics must be noted:

- Only one client can access the 4690 Telnet server at a time.
- When logged on to the 4690 Telnet server, the client shares the main system keyboard and display. This operation is similar to the 4690 Remote Operator capabilities. However, the 4690 Remote Operator and the Telnet server cannot be operated at the same time.

For an alternative Telnet server that does not have these limitations, see “Enhanced Telnet server (ADXHSIUL.286)” on page 36.

Operation

A Telnet entry should be included in the INETD data file, C:\ADX_SDT1\ADXHSIIF.DAT, so that the INETD superserver can start the Telnet server when a client attempts a connection. The example INETD data file copied onto the store controller during 4690 TCP/IP installation contains a Telnet entry. The INETD superserver will start the Telnet server automatically whenever a client issues a connection request.

Logging On and Off

The Telnet server uses the 4690 system user IDs and passwords to grant login access to Telnet clients. For example, the default 4690 system user ID is 99999999 with password of 99999999. When the Telnet client is prompted for user ID and password, the server validates the input against the 4690 system-maintained user IDs and passwords.

To log off normally, enter the Telnet command mode on the client terminal using the Telnet escape sequence (usually Ctrl+]) and type **quit**. Ctrl+D will not close the connection. In addition, Telnet sessions that are idle can be automatically disconnected by the 4690 Telnet server. The timeout for disconnection due to inactivity can be set by creating a user-defined logical file name of ADXTNDTO and assigning it a value (greater than 0) that corresponds to the number of idle minutes the Telnet server will wait before it automatically disconnects. The default timeout is infinite and is used if the ADXTNDTO logical file name does not exist or is defined to an invalid value.

If a client attempts to log on to the server and someone else is already logged on or the server is hung, an option of stopping the active Telnet server process is provided. If you choose to stop the other process, the active session will be disconnected, and you will receive a message telling you to try to log in again.

Using a keyboard

The 4690 System Request function is mapped to the tilde-backquote (~`) key transmitted by the Telnet client. Other function keys transmitted from the PS/2®-style keyboards are known to operate consistently with function keys entered from the 4690 keyboard.

Note: Due to time delays in some network environments, the Telnet server might need to issue more than one read to get an entire function key sequence. For most networks, a 50 ms delay is sufficient for the entire key sequence to arrive. If you are using a WAN environment or a slow network, you might need to define the logical name TNDESCTO to the number of milliseconds you want the Telnet server to wait for an entire function key sequence. The default is 50 ms. Suggested values are 250 to 1000 ms (1/4 to 1 full second).

Locking out the local keyboard: When logged in through the Telnet server, it is possible to lock the local keyboard. This enables only the remotely logged in user to have control of the console. The executable file is ADX_SPGM:ADXHSILL.286. The local keyboard can be locked by issuing the following command:

```
ADXHSILL LOCK
```

The local keyboard will be unlocked upon a normal or abnormal end of the Telnet server, or it can be unlocked with the following command:

```
ADXHSILL UNLOCK
```

Also, you can lock and unlock the local keyboard from any screen without using the ADXHSILL command directly. Typing Ctrl+B blocks the local keyboard and typing Ctrl+U unblocks it.

Terminfo and Termcap files

The 4690 Telnet server does not use TERMINFO or TERMCAP files. There are three basic terminal types supported:

- VT100
- ANSI
- IBM 3151

Also, terminal types of aixterm, xterm, VT100-am (auto margins) are supersets of VT100 and are supported using the VT100 protocols. If an attempt is made to log on to the 4690 Telnet server from an unknown terminal type, the connection will be closed and the client is sent a message indicating that the 4690 Telnet server cannot support the terminal.

Log files

The 4690 Telnet server is actually composed of two processes working together. Program ADXHSIIL.286 is the keyboard input process, and program ADXHSISL.286 is the screen output process. Both of these programs can log messages into the ADXHSIIL.LOG and ADXHSISL.LOG files in the ADX_UPGM subdirectory.

Telnet server considerations

"Real" VT100 terminals are 24-row by 80-column displays. It is recommended that a 25-row VT100 emulator is used. A 24-row emulator can be used, but display results are unpredictable when the 25th row is received (for example, it is sometimes displayed and row 1 scrolls off the display; other times row 25 is not displayed). Some terminal definitions of VT100-am are defined with 25 rows.

Most ANSI terminal types are 25-row by 80-column displays. These terminals work well with the 4690 Telnet server. In addition, if the connection is made to the server from an ANSI terminal type, the color attributes of fields (if you have a color display) will be sent to the terminal. One disadvantage to ANSI terminal types is that there appears to be no consistent definition of function keys and other special keys.

Included on the installation diskette or CD-ROM is an ANSI TERMINFO file for the AIX operating system workstations. The file is called ANSITERM.TI and can be compiled using TIC in the /usr/lib/terminfo directory (you must have root access to do this). You can use this TERMINFO file to define your terminal as ANSI (that is, export TERM=ansi). If your client is on a color display, use ANSI to allow color information to be sent from the 4690 Telnet server. VT100 terminal emulation is black and white only.

Enhanced Telnet server (ADXHSIUL.286)

The 4690 Enhanced Telnet server supports VT220, VT100, ANSI and HFT. It can also support other terminal types, especially where they are similar to the supported types. Unlike the regular Telnet server, the Enhanced server enables multiple clients to be connected simultaneously, and allows you to run the 4690 Remote Operator at the same time. You would configure this through the auxiliary console section of controller configuration.

Operation

A Telnet entry should be included in the INETD data file, C:\ADX_SDT1\ADXHSIIF.DAT, so the INETD superserver can start the Telnet server when a client attempts a connection. The example INETD data file copied onto the store controller during 4690 TCP/IP installation contains the following entry for the regular Telnet server:

```
telnet tcp ADX_SPGM:ADXHSIIL.286
```

This should be changed for the Enhanced Telnet server to:

```
telnet tcp ADX_SPGM:ADXHSIUL.286
```

Note: The INETD superserver will start the Telnet server automatically whenever a client issues a connection request.

Logging On and Off

When a connection is made to the Telnet server, the logon screen is presented, and you must log on as if you were at the main console or another auxiliary console.

To log off normally, enter the Telnet command mode on the client terminal using the Telnet escape sequence (usually **Ctrl+J**) and type **quit**. Alternatively, you can configure a one-key logoff sequence that will log your user ID off normally and terminate your Telnet session. By default, there is no one-key logoff. However, you can change this by defining a user logical name of ADXTNDLO, which contains the ASCII value for the single character you want to use to log off.

For example, use Controller Configuration, User Logical File Names, and define the name ADXTNDLO to have a value of 4. Now when you Telnet into the 4690 store controller, press **Ctrl+D** to log off and terminate your Telnet session, this returns you to where you initiated the session on your Telnet client.

Only the number of sessions configured using Controller Auxiliary Console configuration can be supported. If an additional client attempts to connect when there are already the maximum number of sessions in operation, then the connection is automatically refused.

From the Background Application Control menu you can see a separate copy of ADXHSIUL.286 operating for each connected session. It is also possible to terminate connections from this panel using F8 (Stop).

By default, the Telnet server will request a device status report from the Telnet client after one minute of inactivity. Most clients will automatically respond to this, and the server will keep the connection open. If, after a longer time interval, there is no response, the server will end the connection on the assumption that the client has been removed.

If you want to change the time interval at which this request is sent or disable it altogether, you can define the user logical name ADXTNDTO to the number of minutes you want the interval to be or to 0 minutes to disable it.

Using a keyboard

By default, the backquote character (`) is mapped to the 4690 System Request function. However, you can change this by defining a user logical file name of ADXTNDSR, which contains the ASCII value for the single character you want to map to the 4690 System Request. Also, if you are using a VT220 client, you can use Alt+F12 as an alternate way to generate a 4690 SysRq.

For example, click **Controller Configuration, User Logical File Names**, and define the name ADXTNDSR to have a value of 1. Now, when you Telnet into the 4690 store controller, pressing **Ctrl+A** takes you to the system keys menu.

Note: Due to time delays in some network environments, the Telnet server might need to issue more than one read to get an entire function key sequence. For most networks, a 50 ms delay is sufficient for the entire key sequence to arrive. If you are running over a WAN or a slow network, you might

need to define the logical name TNDESCTO to the number of milliseconds you want the Telnet server to wait for an entire function key sequence. The default is 50 ms. Suggested values are from 250 to 1000 ms (1/4 to 1 full second).

Locking out the local keyboard: The Enhanced Telnet server operates the same as an auxiliary console. It does not lock out the local keyboard.

Terminfo and Termcap files

The 4690 Enhanced Telnet server does not use TERMINFO or TERMCAP files. It has been tested with the following terminal types:

- VT220
- VT100
- ANSI
- HFT

Other terminal types that are similar can also work. The Enhanced Telnet server does not reject connections based upon the terminal type. A valid terminal type must be selected at the client.

Log files

The 4690 Enhanced Telnet server does not log messages to any files. Information messages will be displayed on the background application control panel in the MESSAGE field for the corresponding Telnet server.

Telnet server considerations

Some Telnet clients do not correctly support insert and delete escape sequences. This is most noticeable in 4690 Command Mode, where the output on the client window, when using the delete key, might not accurately reflect what has actually happened on the 4690 system.

See “Telnet server considerations” on page 36 for information about 24-row displays. The OS/2 Warp® Telnet client correctly supports both 25-row displays and the necessary insert and delete escape sequences. Because it supports VT220, you can use Alt+F12 to emulate 4690 SysRq.

The Telnet client in TCP/IP for DOS and Windows 2.1.1.4 works in ANSI mode. However, it does not interpret the insert and delete escape sequences correctly.

LPR client (ADXHSIRL.286)

The LPR client program is invoked from a 4690 command line as ADXHSIRL and accepts the same input parameters as the OS/2 LPR program. All required command parameters can be explicitly provided to the 4690 LPR program. You can define a logical name of LPR_SRV instead of using the “-s <server>” parameter to the LPR program in order to specify the default remote print server. Similarly, you can define a logical name of LPR_PRT instead of using the “-p <printer>” parameter in order to specify the default logical printer on the remote print server.

See the *TCP / IP for 4690 Application Interface Guide* for the command line interface to LPR.

Remote printing through the 4690 print spooler

The LPR client (ADXHSIRL.286) can also be used by the 4690 print spooler mechanism to send print jobs to remote TCP/IP LPD network print servers. When defined to the 4690 system, these network print queues can be accessed in the same way that local 4690 print queues are accessed.

To define a network print queue:

1. Define a network printer on one of the available printer ports, 1 through 8.
2. Select **Installation and Update Aids** from the SYSTEM MAIN MENU.
3. Select **Change Configuration Data** and then **Controller Configuration**.

4. Select **Multiple Printers** and the PRINTER DEFINITION panel displays.
5. Select **Add Printer Definition** for any available printer number and optionally configure this printer as the system printer.
6. Select **Activate Configuration** from the CONFIGURATION MENU to make your changes take effect.
For more detailed information on defining printers, see the *4690 OS: Planning, Installation, and Configuration Guide*.
7. Define the user-defined logical name LPRPARMx, where x is the port number of the previously defined network printer. This logical name contains the parameters taken by the LPR command. This logical name must contain the following two parameters:
 - -S <hostname>, where *hostname* is the name of the LPD server
 - -P <device>, where *device* is the print device on <hostname>
 - -T, <timevalue>, where *timevalue* is the number of seconds that the LPR client should wait for a response from the print server. If the print server does not respond and the timer expires, the LPR client will re-drive the print attempt. This is an optional parameter, but if it is not defined then the LPR Client on the controller will wait indefinitely for a response from the print server.

For a list of valid options to the LPR command, see the *TCP / IP for 4690 Application Interface Guide*.

An example of this is if you have just defined (and activated) a printer on port 7, and you want all print jobs on that queue to be sent to a TCP/IP LPD print server named PRINTSRV, which has a printer attached to device lpt1. You need to define a logical name called LPRPARM7, which is defined to -S PRINTSRV -P LPT1. For more information on user-defined logical names, see the *4690 OS: User's Guide*.

After successfully completing the above configuration steps, you should then be able to access the defined print queue. For example, you can use the command:

```
print <filename> -d=prn7:
```

This command sends the print job to the remote TCP/IP LPD print server.

Note: If you want to use the LPR_SRV or LPR_PRT logical names, as described in “LPR client (ADXHSIRL.286)” on page 38, with the print spooler function, these logical names must be system-level defined logical names. The print spooler will not find process-level logical names.

Rexec client (ADXHSIXL.286)

The Rexec client program is invoked from a 4690 command line as ADXHSIXL and accepts the same input parameters as the OS/2 rexec program. All required command parameters must be explicitly provided to the 4690 rexec program. Enter **REXEC** from the 4690 command line to display a usage message. See the *TCP / IP for 4690 Application Interface Guide* for the command line interface to Rexec.

PCNFSD authentication server

The PCNFSD Authentication Server is required for many PC-based NFS clients. The NFS client must first be authenticated at the server before remote drives/directories can be mounted.

The 4690 PCNFSD Authentication Server can be invoked either from a 4690 command line or as a 4690 background application. It takes no parameters. The ADX_SDT1:PCNFSD.DAT file contains a list of user IDs and passwords that must match those defined on the NFS client in order for authentication to be successful. In 4690 OS Version 2 Release 4 or higher, there are no default passwords and the comment character is a ; (semicolon), which must be in the first position or column one. A sample file is copied to the store controller during installation and can be modified with a text editor.

4690 TCP/IP programming libraries

See the *TCP / IP for 4690 Application Interface Guide* on the Toshiba support site for documentation on the following items. (Select **Publications** under Popular Links).

- Writing 4690 TCP/IP applications
- Compiler, Linker
- Sockets
- FTP API
- SNMP DPI
- SUN RPC

The file names of the 4690 TCP/IP API runtime libraries are listed in Table 6 on page 8. These files can be found on the 4690 Optionals. They are not copied onto the store controller during the Apply Software Maintenance operation. Also, C language header files for developing 4690 TCP/IP applications can be downloaded in a .ZIP format from the supported Web site.

Note: This information describes the 16-bit libraries.

The operating system provides only the socket and RPC library portions of the TCP/IP interfaces for use with the VisualAge compiler. 32-bit versions of the FTP or DPI libraries are not provided. The link library name for the socket interfaces is TCPIP.LIB and the library name for the RPC interfaces is RPC.LIB. The header files used for TCP/IP programming are located in the INCLUDE\TCPIP subdirectory tree of the development environment .ZIP file.

Although *tcperno* is used to retrieve the error values from failed socket calls, it shares its value with the C runtime *errno* value. Therefore, any calls that affect one will affect the other. You must include NERRNO.H to use *tcperno*.

Many function prototypes were missing from the header files shipped with the 16-bit version of TCP/IP. This has been corrected to ensure that all functions can be called properly.

In 16-bit mode, most buffer addresses are checked and an error is returned if a buffer is not addressable by the application. In the 32-bit version of the library, some addresses are checked (at the driver layer) but most are not. Therefore, if an invalid buffer or parameter address is passed in the application will trap (or dump if the dump flag is on).

16-bit to 32-bit conversion

In previous versions of the operating system, the TCP/IP documentation and header files assumed that C integers (type *int*) were 16 bits long. However, with the VisualAge compiler, integers are 32 bits long.

Most of the existing APIs are unchanged (taking and returning integers as arguments). Even though these values are now 32-bits wide, the sockets API is not able to use values larger than the range of a short integer. If lengths larger than the limit of a short integer (32767 or 65535 depending on the function) are passed as arguments to a function an error is returned and *tcperno* is set to EINVAL. The only exception to this rule is the RECV function; the amount of data that can be received is limited to 32767.

Most data structures used with the sockets API have been modified to contain fields of type *short* instead of fields of type *int*. This ensures that the size and layout of these structures are consistent between 16-bit and 32-bit versions. In particular, the *linger* structure used with the `getsockopt()` function is now defined as containing two *shorts* instead of two *ints*.

Retrieving the local host name

The C language `gethostname()` function retrieves the TCP/IP host name of the local host. The syntax of the call is as follows:

```
int gethostname (Name, NameLength)
char *Name;
int NameLength;
```

where:

Name Specifies the address of an array of bytes where the host name is to be stored

NameLength

Specifies the length of the Name array

Upon successful completion, a value of 0 (zero) is returned, and the Name array contains the TCP/IP host name of the local host. If the `gethostname()` function is unsuccessful, a value of -1 is returned.

In order to retrieve the local host name, you must define the user-defined logical name `HOSTNAME`. This logical name contains the fully-qualified TCP/IP host name of the local host. Upon successful completion, the Name array will contain the value of the `HOSTNAME` logical name. If `HOSTNAME` is not defined, `gethostname()` will complete successfully and the Name array will contain "localhost" for the TCP/IP host name. For more information on user-defined logical names, see the *4690 OS: User's Guide*.

TCP/IP in the terminal

The 4690 TCP/IP driver has been enabled for the terminal. When properly configured, a terminal will load the TCP/IP driver and have the capability of communicating via TCP/IP on your network. This enables you to run Java or C language TCP/IP applications on your terminal. Restrictions such as disk dependencies and memory usage apply to these applications as with other terminal applications. Also, `stdin`, `stdout` and `stderr` are redirected to null as with other non-Java terminal applications. 4690 TCP/IP applications shipped for the controller have not been enabled to run on the terminal.

When configuring TCP/IP for the terminal, the only required data entry field is the terminal's IP address. When running TCP/IP applications that require a local host name, such as when using the Java method `getLocalHost()`, you will also need to configure the `HOSTNAME` parameter to retrieve the correct value. Otherwise, a value of "localhost" is returned. Due to space limitations for terminal configuration records, there is an limit of eight characters for the `HOSTNAME` parameter. For more information regarding terminal TCP/IP configuration, see the *4690 OS: Planning, Installation, and Configuration Guide*.

Note: IP addresses can be specified as decimal, hexadecimal, or octal. A leading 0x indicates hex. A leading 0 (zero) indicates octal. Anything else indicates decimal. For example, the following three IP addresses are identical.

- 9.67.39.83
- 0x9.0x43.0x27.0x53
- 011.0103.047.0123

In TCP/IP files that contain IP addresses, any leading zeros in the addresses are interpreted as octal, not decimal. However, in terminal configuration, IP addresses are read as decimal.

Chapter 3. Using IP Security Protocol

This chapter describes using IP Security Protocol (IPsec) in your 4690 operating system. Refer to the *4690 OS: Planning, Installation, and Configuration Guide* for information about installing IPsec.

Notes:

1. The options included in this product can help your company address the PCI DSS requirements.
2. The customer is responsible for evaluation, selection and implementation of security features, administrative procedures and appropriate controls.
3. PCI DSS is Payment Card Industry Data Security Standards.

IPsec overview

IPsec enables secure communications over the Internet and within company networks by securing data traffic at the IP layer. This allows individual users or organizations to secure traffic for all applications without having to make any modifications to the applications. Therefore, the transmission of any data, such as file transfer or application-specific company data, can be made secure.

IP Security and the Operating System

Starting with Version 5, the 4690 operating system uses IPsec, which is an open, standard security technology developed by the Internet Engineering Task Force (IETF). IPsec provides cryptography-based protection of all data at the IP layer of the communications stack. No changes are needed for existing applications. IPsec is the industry-standard network-security framework chosen by the IETF for the IP environment. IPsec protects your data traffic using the following cryptographic techniques:

- **Authentication:** Process by which the identity of a host or end point is verified
- **Integrity checking:** Process of ensuring that no modifications were made to the data while in transit across the network
- **Encryption:** Process of ensuring privacy by *hiding* data and private IP addresses while in transit across the network

Authentication algorithms prove the identity of the sender and data integrity by using a cryptographic hash function to process a packet of data (with the fixed IP header fields included) using a secret key to produce a unique digest. On the receiver side, the data is processed using the same function and key. If either the data has been altered or the sender key is not valid, the datagram is discarded.

Encryption uses a cryptographic algorithm to modify and randomize the data using a certain algorithm and key to produce encrypted data known as *cyphertext*. Encryption makes the data unreadable while in transit. After it is received, the data is recovered using the same algorithm and key (with symmetric encryption algorithms). Encryption must occur with authentication to verify the data integrity of the encrypted data.

These basic services are implemented in IPsec by the use of the Encapsulating Security Payload (ESP) and the Authentication Header (AH). ESP provides confidentiality by encrypting the original IP packet, building an ESP header, and putting the cyphertext in the ESP payload. The AH can be used alone for authentication and integrity-checking if confidentiality is not an issue. With AH, the static fields of the IP header and the data have a hash algorithm applied to compute a keyed digest. The receiver uses its key to compute and compare the digest to make sure the packet is unaltered and the identity of the sender is authenticated.

Two modes of encapsulation are available for IPsec tunnels: transport mode and tunnel mode. In transport mode, the payload of the original IP packet is authenticated and encrypted when using ESP but the IP header is not. AH provides authentication for the entire packet except for certain fields in the IP header that might change in transit. Transport mode is typically used in IPsec tunnels between individual hosts.

With tunnel mode, the original IP packet, including the IP header, is encapsulated in a new IP packet. In this mode, the entire original packet is encrypted and authenticated, providing complete protection. Tunnel mode is required when either end of the tunnel is a gateway (gateways are not required to support transport mode). The advantage of tunnel mode is total protection of the encapsulated IP packet. The disadvantage is extra processing overhead.

Enabling IPsec

IPsec services are loaded during the IPL of a store controller based upon the system configuration selected by the user. Refer to the *4690 OS: Planning, Installation, and Configuration Guide* for information about enabling IPsec during system configuration.

IP Security features

IPsec on 4690 provides the following features:

- AH support using RFC 2402 and ESP support using RFC 2406.
- Tunnel mode and transport mode of encapsulation for host or gateway tunnels.
- Authentication algorithms of HMAC (Hashed Message Authentication Code) MD5 (Message Digest 5) and HMAC SHA (Secure Hash Algorithm).
- Encryption algorithms include 56 bit Data Encryption Standard (DES) Cipher Block Chaining (CBC) with 64 bit initial vector (IV), Triple DES, DES CBC 4 (32 bit IV).
- Filtering of secure and non-secure traffic by a variety of IP characteristics, such as source and destination IP addresses, interface, protocol, port numbers, and more.
- Automatic filter-rule creation and deletion with most tunnel types.
- Use of host names for the source and destination addresses when defining tunnels and filter rules. The host names are converted to IP addresses automatically (as long as a name server or a local HOSTS file or both are available for host name resolution).
- Use of statistics for problem determination.
- User-defined default action which allows the user to specify whether traffic that does not match defined tunnels should be allowed.

IP Security limitations

- IPsec support is only available on 4690 controllers running Classic mode.
- Only manual tunnel support is available on 4690 OS Version 5, which provides static, persistent keys. Internet Key Exchange (IKE) support for automated key management is currently not available.

Tunnels and filters

Two distinct parts of IP Security are *tunnels* and *filters*. Tunnels require filters, but filters do not require tunnels.

Filtering is a function in which incoming and outgoing packets can be accepted or denied based on a variety of characteristics called *rules*. This function allows you to configure the store controller to control the traffic between this controller and other hosts. Filtering is done on a variety of packet properties, such as source and destination addresses, subnet masks, protocol, port, fragmentation, interface, and tunnel definition. This filtering is done at the IP layer, so no changes are required to the applications.

Tunnels define a security association between two hosts. These security associations involve specific security parameters that are shared between end points of the tunnel.

Figure 2 on page 45 indicates the flow of an IPsec packet arriving from the network. The filter driver is called from the IP stack to determine if the packet is permitted or denied. The packet is checked against the existing tunnel definitions and processed according to the matching tunnel definition. If the decapsulation from the tunnel is successful, the packet is passed to the upper-layer protocol. This function

occurs in reverse order for outgoing packets. The tunnel relies on a filter rule to associate the packet with a particular tunnel, but the filtering function can occur without passing the packet to the tunnel. Note that for inbound IPsec packets, traffic passes through filtering twice, once with the IPsec (AH/ESP) headers prior to decapsulation and decryption and a second time, after IPsec processing, as a normal IP packet.

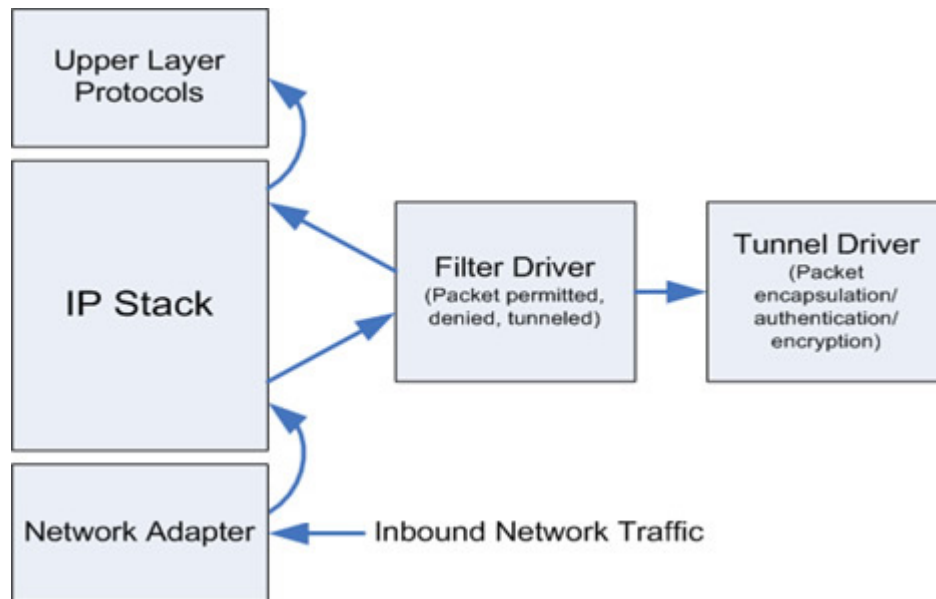


Figure 2. IP Security - Incoming Packet

Tunnels and security associations

Tunnels are used for authenticating, or authenticating and encrypting data. Tunnels are defined by specifying a security association between two hosts. The security association defines the parameters for the encryption and authentication algorithms and characteristics of the tunnel. Figure 3 shows a virtual tunnel between Host A and Host B.

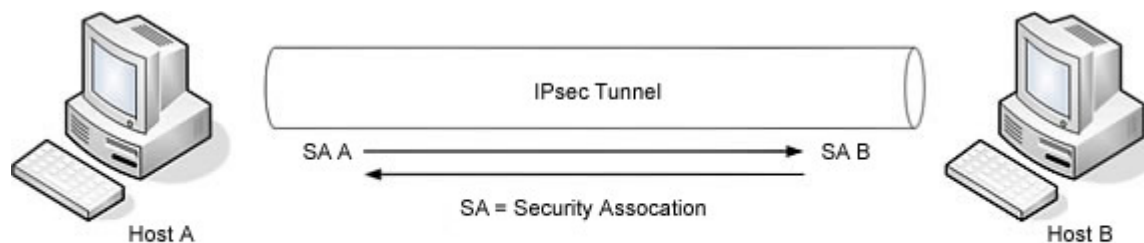


Figure 3. IP Security - Virtual Tunnel

Security association A is indicated by an arrow directed from Host A to Host B. Security association B is indicated by an arrow directed from Host B to Host A. A security association consists of the Destination Address, SPI, Key, Crypto Algorithm and Format, Authentication Algorithm, and Key Lifetime. The Security Parameter Index (SPI) and the destination address identify a unique security association. These parameters are required for uniquely specifying a tunnel. Other parameters such as cryptographic algorithm, authentication algorithm, keys, and lifetime can be specified or defaults can be used.

Filtering capability

Filtering is a basic function in which incoming and outgoing packets can be accepted or denied based on a variety of characteristics. This allows a user or system administrator to configure the host to control the traffic between this host and other hosts. Filtering is done on a variety of packet properties, such as source and destination addresses, subnet masks, protocol, port, fragmentation, interface, and tunnel definition.

Rules, known as *filter rules*, are used to associate certain kinds of traffic with a particular IPsec tunnel. In a basic configuration, when a user defines a host-to-host tunnel, filter rules are auto-generated to direct all traffic from that host through the secure tunnel. If more specific types of traffic are desired, the filter rules can be edited or replaced to allow precise control of the traffic using a particular tunnel. When the tunnel is modified or deleted, the filter rules for that tunnel are automatically modified or deleted, which simplifies IPsec configuration and helps reduce human error.

Filter rules associate particular types of traffic with a tunnel, but data being filtered does not necessarily need to travel in a tunnel. The IPsec function also implements filtering of non-secure packets based on very granular, user-defined criteria. This allows the control of IP traffic between networks and machines that do not require the authentication or encryption properties of IPsec. This aspect of filter rules lets the operating system provide basic firewall functionality to those who want to restrict traffic to or from their machine in an intranet or in a network that does not have the protection of a true firewall. In this scenario, filter rules provide a second barrier of protection around a group of machines. After the filter rules are generated, they are stored in a table and loaded into the driver. When packets are ready to be sent or received from the network, the filter rules are checked in the list from top to bottom to determine whether the packet should be permitted, denied, or sent through a tunnel. The criteria of the rule are compared to the packet characteristics until a match is found or the default rule is reached.

Configuring manual IPsec tunnels

The following procedures are used to configure IPsec tunnels on the 4690 operating system. IPsec tunnels are configured and managed using the following commands:

- `gentun` - create tunnel
- `acttun` - activate tunnel
- `chtun` - change tunnel
- `rmtun` - remove tunnel
- `lstun` - list tunnel

Refer to “IPsec commands” on page 55 for descriptions of these commands.

The process of setting up a tunnel is to define the tunnel on one end, use this tunnel information to set up the definition on the other end, and activate the tunnel and filter rules on both ends. The tunnel is then ready to use.

To set up a manual tunnel, it is not necessary to separately configure the filter rules. If all traffic between two hosts will go through the tunnel, the necessary filter rules can be automatically generated. Settings for the tunnel must be made to match on both sides if they are not explicitly supplied. For instance, the encryption and authentication algorithms specified for the source will be used for the destination if the destination values are not specified.

The following is a sample of the **gentun** command used to create a manual tunnel:

```
gentun -s 10.10.1.2 -d 10.10.1.3 -m tunnel -y Y -N 1000 -n 1001 -p ea -P ea -e  
3DES_CBC -E 3DES_CBC -a HMAC_SHA -A HMAC_SHA
```

The **lstun** command is used to list the characteristics of the manual tunnel created by the previous example. The output looks similar to the following example:

```

Tunnel ID      : 1
Source         : 10.10.1.2
Destination    : 10.10.1.3
Policy        : eaea
Tunnel Mode    : Tunnel
Source AH Algo : HMAC_SHA
Source ESP Algo : 3DES_CBC
Dest AH Algo   : HMAC_SHA
Dest ESP Algo  : 3DES_CBC
Source AH SPI   : 1001
Source ESP SPI  : 1001
Dest AH SPI    : 1000
Dest ESP SPI   : 1000
Tunnel Life Time : 480
Status         : Inactive
Target         : -
Target Mask    : -
Replay        : Yes
Src ESP-Auth Alg : -
Dst ESP-Auth Alg : -

```

To activate the tunnel, type the following command:

```
acttun -t1
```

After it is activated, tunnel 1 is displayed in the active driver list when you enter the command:

```
lstun -a
```

Sample output:

```

Tunnel ID      : 1
Source         : 10.10.1.2
Destination    : 10.10.1.3
Policy        : eaea
Source AH Algo : HMAC_SHA
Source AH SPI   : 1001
Source AH Key   : 0x0009093b0005de7800093897000453fa0005653b
Dest AH Algo   : HMAC_SHA
Dest AH SPI    : 1000
Dest AH Key     : 0x0007a66b000825df000a76d40006551900009665
Source ESP Algo : 3DES_CBC
Source ESP SPI  : 1001
Source ESP Key  : 0x000338970008e5d30006ec000005dbe70008e6df00016c91
Dest ESP Algo  : 3DES_CBC
Dest ESP SPI   : 1000
Dest ESP Key   : 0x00055612000221a00009404b0002a26400082d460004b766
Tunnel Mode    : Tunnel
Replay        : Yes
Reference Count : 0
Deleted       : No
Start         : 10/04/06 17:10:25
Expire        : 10/05/06 01:10:25

```

As shown in the previous example, the AH and ESP keys were automatically generated. When the other tunnel endpoint is configured, the same keys must be used for proper decoding and, therefore, should be entered manually. The **gentun** command provides options for entering keys manually.

The filter rules associated with the tunnel are automatically generated. If this is the first tunnel generated, two predefined filters are generated to permit all AH and ESP traffic for processing IPsec tunnels. For each successive tunnel generated, only the tunnel-specific filters are added. To view the filter rules, use the **lsfilt** command. The output looks similar to the following example, showing the 2 predefined AH and ESP rules and the inbound and outbound tunnel-specific rules.

Rule 1:
Rule action : permit
Source Address : 0.0.0.0
Source Mask : 0.0.0.0
Destination Address : 0.0.0.0
Destination Mask : 0.0.0.0
Protocol : ah
Source Port : any 0
Destination Port : any 0
Direction : both
Fragment control : all packets
Tunnel ID number : 0
Interface : all
Auto-Generated : yes
Expiration Time : 0
Description :

Rule 2:
Rule action : permit
Source Address : 0.0.0.0
Source Mask : 0.0.0.0
Destination Address : 0.0.0.0
Destination Mask : 0.0.0.0
Protocol : esp
Source Port : any 0
Destination Port : any 0
Direction : both
Fragment control : all packets
Tunnel ID number : 0
Interface : all
Auto-Generated : yes
Expiration Time : 0
Description :

Rule 3:
Rule action : permit
Source Address : 10.10.1.2
Source Mask : 255.255.255.255
Destination Address : 10.10.1.3
Destination Mask : 255.255.255.255
Protocol : all
Source Port : any 0
Destination Port : any 0
Direction : outbound
Fragment control : all packets
Tunnel ID number : 1
Interface : all
Auto-Generated : yes
Expiration Time : 0
Description :

Rule 4:
Rule action : permit
Source Address : 10.10.1.3
Source Mask : 255.255.255.255
Destination Address : 10.10.1.2
Destination Mask : 255.255.255.255
Protocol : all
Source Port : any 0
Destination Port : any 0
Direction : inbound
Fragment control : all packets
Tunnel ID number : 1
Interface : all
Auto-Generated : yes
Expiration Time : 0
Description :

Tunnel 1 with its associated filter rules is also activated by the **acttun -t 1** command.

To set up the other side, list the tunnel definition with the **lstun -a** command (print or redirect the output if needed) and use this information as a reference for setting up the algorithm, key, and security parameter index (SPI) values on the other end of the tunnel.

Host-firewall-host configuration

The host-firewall-host configuration option for tunnels allows you to create a tunnel between your host and a firewall, then automatically generate the necessary filter rules for correct communication between your host and a host behind the firewall. The auto-generated filter rules permit all traffic between the two non-firewall hosts over the tunnel specified. The default rules for Authentication Headers (AH) and Encapsulating Security Payload (ESP) headers should already handle the host to firewall communication. The firewall will need to be configured appropriately to complete the setup. Use the tunnel definition created on the host to configure the corresponding SPI values and keys for the firewall endpoint.

Configuring filters

Filtering can be simple, using mostly auto-generated filter rules, or can be customized by defining very specific filter functions based on the properties of the IP packets. Each line in a filter table is known as a *rule*. A collection of rules determine what IP packets are accepted in and out of the machine and how they are directed. Matches to filter rules on incoming IPsec packets are done by comparing the source address and SPI value to those listed in the filter table. Therefore, this pair must be unique. Filter rules can control many aspects of communications, including source and destination addresses and masks, protocol, port number, direction, fragment control, tunnel, and interface type. The types of filter rules are as follows:

- **User-defined filter rules**
- **Auto-generated filter rules**
- **Predefined filter rules**

User-defined filter rules

User-defined filter rules are used in the general filtering of traffic or for associating traffic with IPsec tunnels. They can be added, deleted, modified, and moved. An optional description text field can be added to identify a specific rule.

User-defined filter rules are generated and managed using the following commands:

- **genfilt** – create filter
- **actfilt** – activate filter
- **chfilt** – change filter
- **mvfilt** – move filter
- **rmfilt** – remove filter
- **lsfilt** – list filter

Refer to “IPsec commands” on page 55 for details about these commands.

Auto-generated filter rules

Auto-generated filter rules are a specific set of rules automatically created for the use of IPsec tunnels. For manual tunnels, auto-generated rules specify the source and destination addresses and the mask values, as well as the tunnel ID. All traffic between those addresses will flow through the tunnel. Auto-generated rules permit all traffic over the tunnel. User-defined tunnel rules can place restrictions on certain types of traffic. Place these user-defined tunnel rules before the auto-generated rules, because IPsec uses the first rule it finds that applies to the packet.

To simplify the configuration of a single tunnel, filter rules are auto-generated when tunnels are defined. This function can be suppressed by specifying the **-g** flag in the **gentun** command.

Predefined filter rules

Predefined filter rules are generic filter rules that pertain to all traffic, such as the all traffic or default rule. The default rule cannot be modified, moved, or deleted.

When filters are initialized, a predefined rule (called the default rule) is inserted into the filter table and then activated. By default, this predefined rule is set to permit all packets, but it is user-configurable and can be set to deny all packets. This will keep traffic from passing unless that traffic is specifically defined by additional filter rules.

The ah rule and the esp rules are also predefined rules that allow all esp and ah traffic for IPsec tunnels. The ah and esp rule are auto-generated when the first IPsec tunnel is defined.

Direction

The direction flag (**-w**) of the **genfilt** command specifies whether the rule should be used during input packet processing or output packet processing or both. In IPsec, when filtering is turned on, at least one rule determines the fate of any incoming or outgoing network packet. For example, when a packet is sent out from host A to host B, the outgoing IP packet has the source address of A and the destination address of B. On host A this packet is processed by the filter driver during the outbound processing and on host B during the inbound processing.

Note: The direction **both** implies that the associated rule is used for both incoming and outgoing packets. However, it does not mean that the rule is applied when the source and destination addresses are reversed. For instance, if server A has a rule with *A* as source address and *B* as destination address and the direction is set to **both**, then the rule does not apply on server A to an incoming packet with *B* as the source address and *A* as the destination address. Typically the **both** option is used in gateways that forward the packets.

Subnet masks

Associated with filter rules are *Subnet masks*. Subnet masks are used to group a set of IP addresses that are associated with a filter rule. The mask value is logically ANDed with the IP address in the filter rule (using binary format) and compared to the IP address specified in the packet. For example, a filter rule with a source IP address of 10.10.10.4 and a subnet mask of 255.255.255.255 (binary 11111111.11111111.11111111.11111111) specifies that an exact match must occur of the IP address.

A 10.10.10.x subnet is specified using a subnet mask of 255.255.255.0. An incoming address would have the subnet mask applied to it, then the combination would be compared to the IP address/subnet mask combination in the filter rule. For example, an address of 10.10.10.100 becomes 10.10.10.0 after the subnet mask is applied, which matches the filter rule. A subnet mask of 255.255.255.240 allows any value for the last four bits in the address.

Examples of filter rules

Following is a list of example filters along with descriptions. For readability, the filter list is displayed in a condensed format with space-separated fields. The first list shown provides the name of each field in the filter rule followed by example values from rule 1 in parentheses. The list of example filter rules follows the first list.

- Rule (1)
- Rule Action (permit)
- Source Address (0.0.0.0)
- Source Mask (0.0.0.0)
- Destination Address (0.0.0.0)

- Destination Mask (0.0.0.0)
- Protocol (ah)
- Source Port (source port operator and source port value) (any 0)
- Destination Port (destination port operator and destination port value (any 0)
- Direction (both)
- Fragment Control (all packets)
- Tunnel ID number (0)
- Interface (all)

The following is a list of example filter rules:

```

1 permit 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 ah any 0 any 0 both all packets 0 all
0 none none

2 permit 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 esp any 0 any 0 both all packets 0 all
0 none none

3 permit 10.10.1.2 255.255.255.255 10.10.1.3 255.255.255.255 all any 0 any 0
outbound all packets 1 all 0 none none

4 permit 10.10.1.3 255.255.255.255 10.10.1.2 255.255.255.255 all any 0 any 0
inbound all packets 1 all 0 none none

5 permit 10.10.1.2 255.255.255.255 10.10.1.3 255.255.255.255 icmp any 0 any 0
outbound all packets 2 all 0 none none

6 permit 10.10.1.3 255.255.255.255 10.10.1.2 255.255.255.255 icmp any 0 any 0
inbound all packets 2 all 0 none none

7 permit 10.10.1.2 255.255.255.255 10.10.1.5 255.255.255.255 tcp gt 1023 eq 21
outbound all packets 3 all 0 none none

8 permit 10.10.1.5 255.255.255.255 10.10.1.2 255.255.255.255 tcp/ack eq 21 gt
1023 inbound all packets 3 all 0 none none

9 permit 10.10.1.5 255.255.255.255 10.10.1.2 255.255.255.255 tcp eq 20 gt 1023
inbound all packets 3 all 0 none none

10 permit 10.10.1.2 255.255.255.255 10.10.1.5 255.255.255.255 tcp/ack gt 1023
eq 20 outbound all packets 3 all 0 none none

11 permit 10.10.1.2 255.255.255.255 10.10.1.5 255.255.255.255 tcp gt 1023 gt
1023 outbound all packets 3 all 0 none none

12 permit 10.10.1.5 255.255.255.255 10.10.1.2 255.255.255.255 tcp/ack eq 1023
gt 1023 inbound all packets 3 all 0 none none

0 permit 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0 all any 0 any 0 both all packets 0 all
0 none none Default Rule

```

Each rule in the previous example is described in Table 8.

Table 8. Explanation of filter rules example

Rule	Explanation
Rules 1 and 2	Auto-generated rules that allow processing of authentication headers (AH) and encapsulating security payload (ESP) headers for IPsec tunnels. Note: Do not modify Rules 1 and 2.
Rules 3 and 4	Set of auto-generated rules that filter traffic between addresses 10.10.1.2 and 10.10.1.3 through tunnel 1. Rule 3 is for outbound traffic and rule 4 is for inbound traffic.

Table 8. Explanation of filter rules example (continued)

Rule	Explanation
Rules 5 and 6	<p>Set of user-defined rules that filter both inbound and outbound services of any type between addresses 10.10.1.2 and 10.10.1.3 through tunnel 2.</p> <p>Commands to generate:</p> <pre>genfilt -s 10.10.1.2 -d 10.10.1.3 -a P -c icmp -t 2 -w 0 genfilt -s 10.10.1.3 -d 10.10.1.2 -a P -c icmp -t 2 -w I</pre>
Rules 7 through 12	<p>User-defined filter rules that filter outbound file transfer protocol (FTP) service from 10.10.1.2 and 10.10.1.5 through tunnel 3.</p> <p>Commands to generate:</p> <pre>genfilt -s 10.10.1.2 -d 10.10.1.5 -a P -c tcp -o gt -p 1023 -0 eq -P 21 -t 3 -w 0 genfilt -s 10.10.1.5 -d 10.10.1.2 -a P -c tcp/ack -o eq -p 21 -0 gt -P 1023 -t 3 -w I genfilt -s 10.10.1.5 -d 10.10.1.2 -a P -c tcp -o eq -p 20 -0 gt -P 1023 -t 3 -w I genfilt -s 10.10.1.2 -d 10.10.1.5 -a P -c tcp/ack -o gt -p 1023 -0 eq -P 20 -t 3 -w 0 genfilt -s 10.10.1.2 -d 10.10.1.5 -a P -c tcp -o gt -p 1023 -0 gt -P 1023 -t 3 -w 0</pre>
Rule 0	<p>Predefined rule, known as the default rule, which is always placed at the end of the table. In this example, it permits all packets that do not match the other filter rules. It can be set to deny all traffic not matching the other filter rules.</p>

Each rule can be viewed separately (using the **lsfilt** command) to list each field with its value. For example, entering **lsfilt -n 7** shows the following (**lsfilt** with no parameters would list all the rules in this format):

```
Rule 7:
Rule action      : permit
Source Address   : 10.10.1.2
Source Mask      : 255.255.255.255
Destination Address : 10.10.1.5
Destination Mask  : 255.255.255.255
Protocol         : tcp
Source Port      : gt 1023
Destination Port  : eq 21
Direction       : outbound
Fragment control : all packets
Tunnel ID number : 3
Interface       : all
Auto-Generated   : no
Expiration Time  : 0
Description      :
```

Filters and intrusion detection and prevention

Intrusion detection and prevention is the action of monitoring and analyzing system events in order to intercept and reject any attempt of unauthorized system access. In 4690, this detection and prevention of unauthorized access or attempted unauthorized access is done by observing certain actions, and then applying filter rules to these actions.

Pattern matching filter rules

Pattern matching is the use of an IPsec filter rule for filtering networking packets. A filter pattern can be a text string, a hexadecimal string, or a file containing more than one pattern. After a pattern filter rule is established and that pattern is detected in the body of any network packet, then the predefined action of the filter rule will result.

Pattern matching only applies to inbound network packets. Use the **genfilt** command to add a pattern matching filter rule to the filter rule table. Use the **actfilt** command to activate or deactivate the filter rules. A pattern file can contain a list, one per line, of text patterns or hexadecimal patterns. Pattern matching filter rules can be used to guard against viruses, buffer overflows, and other network security attacks.

Pattern matching filter rules can have a negative impact on system performance if they are used too broadly and with a high number of patterns. It is best to keep the scope of their application as narrow as possible. For example, if a known virus pattern applies to a specific application, then specify the destination port of the application in the filter rule. This allows all other traffic to pass without incurring a performance impact from pattern matching.

Note: Pattern matching is performed on incoming packets only.

There are three basic types of patterns: text, hexadecimal, and file.

- **Text pattern** — A text filter pattern is an ASCII string that looks similar to the following: GET
/../../../../../../../../
- **Hexadecimal pattern** — A hexadecimal pattern looks similar to the following:
0x33c0b805e0cd16b807e0cd1650558becc7460200f05d0733ffb8c800b9ffff3abb
00150e670e47132c0e67158fec03c80

Note: A hexadecimal pattern is differentiated from a text pattern by the leading 0x.

- **Files that contain patterns** — A file can contain a list, one per line, of text patterns or hexadecimal patterns.

Shun port and shun host filter rules

By setting a shun filter rule, you can prevent a remote host or the remote host and port pair from accessing the local machine. A shun filter rule dynamically creates an effect rule that denies the remote host or the remote host and port pair from accessing the local machine when the specified criteria of the rule are met. Because it is common for an attack to be preceded by a port scan, shun filter rules are especially useful in preventing an intrusion by detecting this attack behavior.

For example, if the local host does not use the server port 37, which is the time server, then the remote host should not be accessing port 37, unless it is running a port scan. Place a shun filter rule on port 37 so that if the remote host attempts to access that port, the shun filter rule creates an effective rule that blocks that host from further access for the amount of time specified in the shun rule **expiration time** field. If the **expiration time** field of a shun rule is set to 0, then the dynamically created effective shun rule does not expire.

Notes:

1. An expiration time specified by the shun port or shun host filter rule applies only to the dynamically created effect rule.
2. Dynamically created effect rules can only be viewed with the **lsfilt -a** command.
3. When the criteria of a shun host filter rule is met, then the dynamically created effective rule will block or shun all network traffic from the remote host for the specified expiration time.
4. When the criteria of a shun port filter rule is met, then the dynamically created effective rule will only block or shun network traffic from this remote host's particular port, until the expiration time is exceeded.

Shun host example: The following example shows a shun host rule to shun any host for 2 minutes on the 10.10.1.0 network that attempts an FTP request. The command to generate the rule is as follows:

```
genfilt -a H -s 10.10.1.0 -m 255.255.255.0 -d 10.10.1.2 -c tcp -O eq -P 21 -e 120 -D  
"shun host on FTP attempt for 2 min"
```

The **lsift** command would show the following:

Beginning of filter rules.

Rule 1:

```
Rule action      : shunHost
Source Address   : 10.10.1.0
Source Mask      : 255.255.255.0
Destination Address : 10.10.1.2
Destination Mask  : 255.255.255.255
Protocol         : tcp
Source Port      : any 0
Destination Port  : eq 21
Direction       : both
Fragment control : all packets
Tunnel ID number : 0
Interface        : all
Auto-Generated   : no
Expiration Time   : 120
Description      : shun host on FTP attempt for 2 min
```

Rule 0:

```
Rule action      : permit
Source Address   : 0.0.0.0
Source Mask      : 0.0.0.0
Destination Address : 0.0.0.0
Destination Mask  : 0.0.0.0
Protocol         : all
Source Port      : any 0
Destination Port  : any 0
Direction       : both
Fragment control : all packets
Tunnel ID number : 0
Interface        : all
Auto-Generated   : no
Expiration Time   : 0
Description      : Default Rule
```

End of filter rules

When host 10.10.1.3 attempts an FTP to host 10.10.1.2, a dynamically created effective rule appears in the active filter list (**lsfilt -a**) to deny all traffic from host 10.10.1.3 for 2 minutes. The expiration time shown in the filter rule is approximate as indicated by the tilde (~). When the expiration time is reached, the rule is automatically deleted from the active list as shown in the following example.

Beginning of filter rules.

Rule 1:

```
Rule action      : deny
Source Address   : 10.10.1.3
Source Mask      : 255.255.255.255
Destination Address : 0.0.0.0
Destination Mask  : 0.0.0.0
Protocol         : all
Source Port      : any 0
Destination Port  : any 0
Direction       : both
Fragment control : all packets
Tunnel ID number : 0
Interface        : all
~Expiration Time : 92
```

Rule 2:

```
Rule action      : shunHost
Source Address   : 10.10.1.0
Source Mask      : 255.255.255.0
Destination Address : 10.10.1.2
Destination Mask  : 255.255.255.255
Protocol         : tcp
Source Port      : any 0
Destination Port  : eq 21
```

```
Direction      : both
Fragment control : all packets
Tunnel ID number : 0
Interface      : all
Expiration Time : 120
```

Rule 3:

```
Rule action      : permit
Source Address   : 0.0.0.0
Source Mask      : 0.0.0.0
Destination Address : 0.0.0.0
Destination Mask : 0.0.0.0
Protocol         : all
Source Port      : any 0
Destination Port : any 0
Direction       : both
Fragment control : all packets
Tunnel ID number : 0
Interface        : all
Expiration Time  : 0
```

End of filter rules.

Stateful filter rules

Stateful filters examine information such as source and destination addresses, port numbers, and status. Then, by applying IF, ELSE or ENDIF filter rules to these header flags, stateful systems can make filtering decisions in the context of an entire session rather than that of an individual packet and its header information.

Stateful inspection examines incoming and outgoing communication packets. When stateful filter rules are activated with the **actfilt -u** command, the rules in the ELSE block are always examined until the IF rule is satisfied. After the IF rule or condition is satisfied, the rules in the IF block are used until the filter rules are reactivated with the **actfilt -u** command.

Timed rules

Timed rules specify the amount of time, in seconds, that the filter rule is applied after it is made effective with the **actfilt -u** command. The expiration time is specified with the **-e** option of the **genfilt** command. For more information, see “actfilt command (adxipsaf.386)” on page 58 and “genfilt command (adxipsaf.386).”

Note: Timers have no effect on IF, ELSE or ENDIF rules. If an expiration time is specified in a shun host or shun port rule, the time applies only to the effect rule that is initiated by the shun rule. Shun rules have no expiration time.

IPsec commands

Manual IPsec tunnels and filters are configured, activated, managed, and displayed using command line functions. These command line functions save the tunnel and filter records in configuration files and pass the configuration to the appropriate drivers by means of special calls. The following sections describe the commands.

genfilt command (adxipsaf.386)

This section describes the **genfilt** command.

Purpose

Adds a filter rule to the filter configuration database.

Syntax

genfilt [**-n** *fid*] [**-a** **D|P|I|L|E|H|S**] **-s** *s_addr* **-m** *s_mask* [**-d** *d_addr*] [**-M** *d_mask*] [**-c** *protocol*] [**-o** *s_opr*] [**-p** *s_port*] [**-O** *d_opr*] [**-P** *d_port*] [**-w** **I|O|B**] [**-f** **Y|N|O|H**] [**-t** *tid*] [**-i** *interface*] [**-D** *description*] [**-e** *expiration_time*] [**-x** *quoted_pattern*] [**-X** *pattern_filename*]

Description

Use the command **genfilt** to add a filter rule to the filter configuration database. The filter rules generated by this command are called user-defined filter rules.

Flags

-a *action*

The following action values are allowed:

- **D** (deny) blocks traffic.
- **P** (permit) allows traffic.
- **I** (if) makes this an IF filter rule.
- **L** (else) makes this an ELSE filter rule.
- **E** (endif) makes this an ENDIF filter rule.
- **H** (shunHost) makes this a SHUN HOST filter rule.
- **S** (shunPort) makes this a SHUN PORT filter rule.

Permit is the default action, if no other action is specified.

All IF rules must be closed with an associated ENDIF rule. These conditional rules can be nested, but correct nesting and scope must be adhered to or the rules will not load correctly with the **actfilt** command.

-c *protocol*

Specifies the protocol. The valid values are: **udp**, **icmp**, **tcp**, **tcp/ack**, **esp**, **ah**, and **all**. The value **all** indicates that the filter rule will apply to all the protocols. The protocol can also be specified numerically (between 1 and 252). The default value is **all**.

-D *description*

A short text description for the filter rule, specified as a quoted character string. This is an optional flag for user-defined filter rules. Descriptions are truncated at 80 characters.

-d *a_addr*

Specifies the destination address. It can be an IP address or a host name. If a host name is specified, the first IP address returned by the name server for that host will be used. This value, along with the destination subnet mask, will be compared against the destination address of the IP packets. If unspecified, the destination address defaults to 0.0.0.0 (indicating "any" address).

-e *expiration_time*

Specifies the expiration time. The expiration time is the amount of time the rule should remain active in seconds. The *expiration_time* does not remove the filter rule from the database. The *expiration_time* relates to the amount of time the filter rule is active while processing network traffic. If no *expiration_time* is specified, then the life time of the filter rule is infinite. If *expiration_time* is specified in conjunction with a SHUN PORT (**-a S**) or SHUN HOST (**-a H**) filter rule, then this is the amount of time the remote port or remote host is denied or shunned when the filter rule parameters are met. If *expiration_time* is specified independent of a shun rule, then this value is the amount of time the filter rule will remain active after the filter rules are loaded into the driver and start processing network traffic. The valid values for *expiration_time* are 0 – 2592000 seconds, where 0 represents no expiration.

-f **Y|N|O|H**

Specifies the fragmentation control. This flag specifies that this rule will apply to either all packets (**Y**), fragment headers and unfragmented packets only (**H**), fragments and fragment headers only (**O**), or unfragmented packets only (**N**). The default value is **Y**.

-i interface

Specifies the name of IP interface(s) to which the filter rule applies. The examples of the interface name are: **all**, **tr0**, and **en0**. The default value is **all**.

-m s_mask

Specifies the source subnet mask. This is used in the comparison of the source address of the IP packet with the source address of the filter rule. This parameter is required if **-d** is not specified. If unspecified, the subnet mask defaults to 255.255.255.255, indicating an exact match with the specified source address, unless the source address is 0.0.0.0. An IP address of 0.0.0.0 defaults to a mask of 0.0.0.0 for correct operation of the filter.

-M d_mask

Specifies the destination subnet mask. This is used in the comparison of the destination address of the IP packet with the destination address of the filter rule. If unspecified, the subnet mask defaults to 255.255.255.255, indicating an exact match with the specified destination address, unless the destination address is 0.0.0.0. An IP address of 0.0.0.0 defaults to a mask of 0.0.0.0 for correct operation of the filter.

-n fid Specifies the filter rule ID for filter rule placement. The new rule will be added BEFORE the specified filter rule. The ID must be greater than 0 and less than the last filter ID +1, as shown by the **lsfilt** command. If this flag is not used, the new rule will be added to the end of the filter rule table.

-o s_opr

Specifies the source port or ICMP type operation. This is the operation that will be used in the comparison between the source port or ICMP type of the packet and the source port or ICMP type (**-p** flag) specified in this filter rule. The valid values are: **lt** (less than), **le** (less than or equal), **gt** (greater than), **ge** (greater than or equal), **eq** (equal), **neq** (not equal), and **any**. The default value is **any**.

-O d_opr

Specifies the destination port or ICMP code operation. This is the operation that will be used in the comparison between the destination port or ICMP code of the packet and the destination port or ICMP code (**-P** flag) specified in this filter rule. The valid values are: **lt** (less than), **le** (less than or equal), **gt** (greater than), **ge** (greater than or equal), **eq** (equal), **neq** (not equal), and **any**. The default value is **any**.

-p s_port

Specifies the source port or ICMP type. This is the value or ICMP type that will be compared to the source port or ICMP type of the IP packet. The **-o** option must be specified when the **-p** option is used.

-P d_port

Specifies the destination port or ICMP code. This is the value or ICMP code that will be compared to the destination port or ICMP code of the IP packet. The **-O** option must be specified when the **-P** option is used.

-s s_addr

Specifies the source address. It can be an IP address or a host name. If a host name is specified, the first IP address returned by the name server for that host will be used. This value, along with the source subnet mask, will be compared against the source address of the IP packets.

-t tid Specifies the ID of the tunnel related to this filter rule. All the packets that match this filter rule must go through the specified tunnel. If this flag is not specified, the value is 0 which indicates that this rule only applies to non-tunnel traffic.

-w direction

Specifies whether the rule applies to incoming packets (**I**), outgoing packets (**O**), or both (**B**). The default value is **B**. It is not valid to use **O** (outgoing packets) with the **-x** or **-X** pattern options. It is valid to specify the (**B**) both directions with the pattern options, but only the incoming packets are checked against the patterns.

-x *quoted_pattern*

Specifies a pattern to compare against network traffic. The pattern can be specified as a quoted ASCII character string or a hexadecimal pattern preceded by 0x. The pattern is a minimum of 4 and a maximum of 2048 ASCII characters or hexadecimal bytes. Hexadecimal patterns must be specified as full bytes (2 hexadecimal characters per byte). For long patterns, use the **-X** option with a pattern file because of the command line size limitation.

-X *pattern_filename*

Specifies the pattern file name. If more than one pattern is associated with this filter rule, then a pattern file name must be used. The pattern file must be in the format of one pattern per line. A pattern is an unquoted character string, interpreted as an ASCII string unless preceded by 0x, in which case it is interpreted as a hexadecimal pattern. Each pattern is a minimum of 4 and a maximum of 2048 ASCII characters or hexadecimal bytes. Each pattern in the file will be truncated at 2048 ASCII characters or hexadecimal bytes. Hexadecimal patterns must be specified as full bytes (2 hexadecimal characters per byte). This file is read when the filter rules are activated.

Note: The 4690 XE editor has a limit of approximately 1024 characters per line; therefore, pattern files with longer patterns should be created on a different system and transferred to the controller.

actfilt command (adxipsaf.386)

This section describes the **actfilt** command.

Purpose

Activates or deactivates the filter rules.

Syntax

actfilt -dlu [**-z PID**] [**-i**]

Description

Use the **actfilt** command to activate or deactivate the filter rules. This command activates or deactivates the filter rules in the active driver filter table. After being activated, filters will be matched with incoming and outgoing packets. IPsec filter rules for this command are configured using the **genfilt** command.

Flags

- d** Deactivates the active filter rules. This flag cannot be used with the **-u** flag.
- u** Activates the filter rules in the filter rule table. This flag cannot be used with the **-d** flag.
- z PID** Sets the action of the default filter rule to “permit” (**P**) or “deny” (**D**). The default filter rule is the last rule in the filter rule table that will apply to traffic that does not apply to any other filter rules in the table. Setting the action of this rule to “permit” will allow all traffic that does not apply to any other filter rules. Setting this action to “deny” will not allow traffic that does not apply to any other filter rules.
- i** Initialization flag. This flag only applies when the **-u** flag is also used. If the **-i** flag is used, all the filter rules with an “active” status will be activated. If not used, all the filter rules in the filter rule table will be activated. The **actfilt -u -i** command is issued during IPL to activate the filter rules that were active just before the IPL.

chfilt command (adxipscf.386)

This section describes the **chfilt** command.

Purpose

Changes a filter rule.

Syntax

chfilt -n *fid* [-a **D|P|I|L|E|H|S**] [-s *s_addr*] [-m *s_mask*] [-d *d_addr*] [-M *d_mask*] [-c *protocol*] [-o *s_opr*] [-p *s_port*] [-O *d_opr*] [-P *d_port*] [-w **I|O|B**] [-f **Y|N|O|H**] [-t *tid*] [-i *interface*] [-D *description*] [-e *expiration_time*] [-x *quoted_pattern*] [-X *pattern_filename*]

Description

Use the **chfilt** command to change the definition of a filter rule in the filter rule table. Specify one or more parameters to change in the filter specified by the -n parameter. All other filter parameters remain as previously configured by the **genfilt** command. Auto-generated filter rules and manual filter rules can be changed by this command. If an auto-generated filter rule is modified by the **chfilt** command, it becomes a manual filter rule. IPsec filter rules for this command are configured using the **genfilt** command.

Flags

-a *action*

The following action values are allowed:

- **D** (deny) blocks traffic.
- **P** (permit) allows traffic.
- **I** (if) makes this an IF filter rule.
- **L** (else) makes this an ELSE filter rule.
- **E** (endif) makes this an ENDIF filter rule.
- **H** (shunHost) makes this a SHUN HOST filter rule.
- **S** (shunPort) makes this a SHUN PORT filter rule.

All IF rules must be closed with an associated ENDIF rule. These conditional rules can be nested, but correct nesting and scope must be adhered to or the rules will not load correctly with the **actfilt** command.

-c *protocol*

Specifies the protocol. The valid values are: **udp**, **icmp**, **tcp**, **tcp/ack**, **esp**, **ah**, and **all**. The value **all** indicates that the filter rule will apply to all the protocols. The protocol can also be specified numerically (between 1 and 252).

-D *description*

A short text description for the filter rule, specified as a quoted character string. This is an optional flag for user-defined filter rules. Descriptions are truncated at 80 characters.

-d *a_addr*

Specifies the destination address. It can be an IP address or a host name. If a host name is specified, the first IP address returned by the name server for that host will be used. This value, along with the destination subnet mask, will be compared against the destination address of the IP packets.

-e *expiration_time*

Specifies the expiration time. The expiration time is the amount of time the rule should remain active in seconds. The *expiration_time* does not remove the filter rule from the database. The *expiration_time* relates to the amount of time the filter rule is active while processing network traffic. If no *expiration_time* is specified, then the life time of the filter rule is infinite. If the *expiration_time* is specified in conjunction with a SHUN PORT (-a **S**) or SHUN HOST (-a **H**) filter rule, then this is the amount of time the remote port or remote host is denied or shunned when the filter rule parameters are met. If the *expiration_time* is specified independent of a shun rule, then this value is the amount of time the filter rule will remain active after the filter rules are loaded into the kernel and start processing network traffic. The valid values for *expiration_time* are 0 – 2592000 seconds, where 0 represents no expiration.

-f YINIOIH

Specifies the fragmentation control. This flag specifies that this rule will apply to either all packets (**Y**), fragment headers and unfragmented packets only (**H**), fragments and fragment headers only (**O**), or unfragmented packets only (**N**).

-i interface

Specifies the name of IP interfaces to which the filter rule applies. The examples of the interface name are: **all**, **tr0**, and **en0**.

-m s_mask

Specifies the source subnet mask. This mask will be applied to the source address (**-s** flag) when compared with the source address of the IP packet.

-M d_mask

Specifies the destination subnet mask. This mask will be applied to the destination address (**-d** flag) when compared with the destination address of the IP packet.

-n fid Specifies the ID of the filter rule to change. It must exist in the filter rule table and it cannot be 0. (Rule 0 is the default filter rule and cannot be changed with **chfilt** command).

-o s_opr

Specifies the source port or ICMP type operation. This is the operation that will be used in the comparison between the source port or ICMP type of the packet and the source port or ICMP type (**-p** flag) specified in this filter rule. The valid values are: **lt** (less than), **le** (less than or equal), **gt** (greater than), **ge** (greater than or equal), **eq** (equal), **neq** (not equal), and **any**. The **-o** option must be specified when the **-p** option is used.

-O d_opr

Specifies the destination port or ICMP code operation. This is the operation that will be used in the comparison between the destination port or ICMP code of the packet and the destination port or ICMP code (**-P** flag) specified in this filter rule. The valid values are: **lt** (less than), **le** (less than or equal), **gt** (greater than), **ge** (greater than or equal), **eq** (equal), **neq** (not equal), and **any**. The **-O** option must be specified when the **-P** option is used.

-p s_port

Specifies the source port or ICMP type. This is the value or ICMP type that will be compared to the source port or ICMP type of the IP packet.

-P d_port

Specifies the destination port or ICMP code. This is the value or ICMP code that will be compared to the destination port or ICMP code of the IP packet.

-s s_addr

Specifies the source address. It can be an IP address or a host name. If a host name is specified, the first IP address returned by the name server for that host will be used. This value, along with the source subnet mask, will be compared against the source address of the IP packets.

-t tid Specifies the ID of the tunnel related to this filter rule. All the packets that match this filter rule must go through the specified tunnel. A value of 0 indicates that no tunnels are associated with this filter.

-w direction

Specifies whether the rule applies to incoming packets (**I**), outgoing packets (**O**), or both (**B**). It is not valid to use **O** (outgoing packets) with the **-x** or **-X** pattern options. It is valid to specify the (**B**) both directions with the pattern options, but only the incoming packets are checked against the patterns.

-x quoted_pattern

Specifies a pattern to compare against network traffic. The pattern can be specified as a quoted ASCII character string or a hexadecimal pattern preceded by 0x. The pattern is a minimum of 4 and a maximum of 2048 ASCII characters or hexadecimal bytes. Hexadecimal patterns must be

specified as full bytes (2 hexadecimal characters per byte). For long patterns, use the **-X** option with a pattern file because of command line size limitations.

-X *pattern_filename*

Specifies the pattern file name. If more than one pattern is associated with this filter rule, then a pattern file name must be used. The pattern file must be in the format of one pattern per line. A pattern is an unquoted character string, interpreted as an ASCII string unless preceded by 0x, in which case it is interpreted as a hexadecimal pattern. The pattern is a minimum of 4 and a maximum of 2048 ASCII characters or hexadecimal bytes. Hexadecimal patterns must be specified as full bytes (2 hexadecimal characters per byte). This file is read when the filter rules are activated.

mvfilt command (adxipsmf.386)

This section describes the **mvfilt** command.

Purpose

Moves a filter rule.

Syntax

mvfilt -p *p_fid* -n *n_fid*

Description

Use the **mvfilt** command to change the position of a filter rule in the filter rule table. IPsec filter rules for this command are configured using the **genfilt** command.

Flags

-p *p_fid*

Filter rule ID. It specifies the previous position of the rule in the filter rule table. The values of 0 and the last filter position in the filter table are not valid since the default filter rule is unmovable and becomes the last filter rule when activated.

-n *n_fid*

Filter rule ID. It specifies the new position of the rule in the filter rule table after the move. The values of 0 and the last filter position in the filter table are not valid since the default filter rule is unmovable and becomes the last filter rule when activated.

rmfilt command (adxipsrf.386)

This section describes the **rmfilt** command.

Purpose

Removes a filter rule from the filter rule table.

Syntax

rmfilt -n *fid* | all [-f] [-q]

Description

Use the **rmfilt** command to remove filter rules from the filter rule table. Actions by this command will not affect the active driver filter list until the **actfilt** command is executed. IPsec filter rules for this command are configured using the **genfilt** command. Only manual filter rules can be removed.

Flags

-n *fid* | all

The ID of the filter rule to remove from the filter rule table. The value of 0 is not valid for this flag since it is the default filter rule. If **all** is specified, all the user-defined filter rules will be removed.

- f** Force to remove auto-generated filter rules. The **-f** flag works with **-n all** to remove all the filter rules (user-defined and auto-generated filter rules) except rule number 0.
- q** The **-q** flag indicates “quiet mode” and suppresses the per-filter informational messages.

Isfilt command (adxipself.386)

This section describes the **Isfilt** command.

Purpose

Lists filter rules from either the filter table in the filter configuration database or the active driver filter list.

Syntax

Isfilt [-n *fid_list*] [-a]

Description

Use the **Isfilt** command to list filter rules and their status. **Isfilt** displays filters from the filter configuration database (without **-a**) or from the active driver filter list (with **-a**). The default filter rule is always displayed last when listing filter rules since it is always the last rule applied to filter network traffic. The default filter rule ID is 0 in the filter configuration database and, when activated, it becomes the last or highest filter ID in the filter table.

Note: Filter description fields are not listed in the active driver filter list. No filter description text will be displayed when active rules are listed.

Flags

- a** List only the active filter rules. The active filter rules are rules that are currently in use by the filter driver. If **-a** is omitted, all the filter rules in the filter rule table will be listed.
- n *fid_list***
Specifies the IDs of filter rules to be displayed. The *fid_list* is a single filter ID or a list of filter IDs separated by a space, a comma (,) or a dash (-), where “-” represents a range. Filter ID 0 (the default rule) cannot be specified in a range, but can be listed separately. For example, **Isfilt -n 1-3,0** will list filter rules 1,2,3, and 0. The **-n** option is not valid for active filter rules. This flag cannot be used with the **-a** flag.

gentun command (adxipsgt.386)

This section describes the **gentun** command.

Purpose

Creates a tunnel definition in the tunnel database.

Syntax

gentun -s *src_host_IP_address* -d *dst_host_IP_address* [-m *tunnel | transport*] [-e [*src_esp_algo*]] [-a [*src_ah_algo*]] [-b *src_esp_auth_algo*] [-p *src_policy*] [-E *dst_esp_algo*] [-A [*dst_ah_algo*]] [-B *dst_esp_auth_algo*] [-P *dst_policy*] [-k *src_esp_key*] [-h *src_ah_key*] [-c *src_esp_auth_key*] [-K *dst_esp_key*] [-H *dst_ah_key*] [-C *dst_esp_auth_key*] [-n *src_esp_spi*] [-u *src_ah_spi*] [-N *dst_esp_spi*] [-U *dst_ah_spi*] [-l *lifetime*] [-y *YIN*] [-f *fw_address*] [-x *dst_mask*]] [-g]

Description

The **gentun** command creates a definition of a tunnel between a local host and a tunnel partner host. The associated auto-generated filter rules for the tunnel can be optionally generated by this command.

Flags

- a *src_ah_algo***
Source AH authentication algorithm, used by source for IP packet authentication. The list of all the authentication algorithms is shown in Table 9 on page 65. The default value is **HMAC_MD5**.

- A *dst_ah_algo***
Destination AH Authentication Algorithm, used by destination for IP packet authentication. The list of all the authentication algorithms is shown in Table 9 on page 65. If this flag is not used, the value used by the **-a** flag is used.
- b *src_esp_auth_algo***
Source ESP Authentication Algorithm. The list of all the authentication algorithms is shown in Table 9 on page 65.
- B *dst_esp_auth_algo***
Destination ESP Authentication Algorithm. The list of all the authentication algorithms is shown in Table 9 on page 65. If this flag is not used, it is set to the same value as the **-b** flag.
- c *src_esp_auth_key***
Source ESP Authentication Key. It must be a hexadecimal string started with 0x. If this flag is not used, the system will generate one for you.
- C *dst_esp_auth_key***
Destination ESP Authentication Key. It must be a hexadecimal string started with 0x. If this flag is not used, it is set to the same value as the **-c** flag.
- d *dst_host_IP_address***
Destination Host IP address. In the host-to-host case, this is the IP address of the destination host interface to be used by the tunnel. In the host-firewall-host case, this is the IP address of the destination host behind the firewall. A host name is also valid and the first IP address returned by the name server for the host name will be used.
- e *src_esp_algo***
Source ESP Encryption algorithm, used by the source for IP packet encryption. The list of all the encryption algorithms is shown in Table 9 on page 65.
- E *dst_esp_algo***
Destination ESP Encryption algorithm, used by destination for IP packet encryption. The list of all the encryption algorithms is shown in Table 9 on page 65. If this flag is not used, the value used by the **-e** flag is used.
- f *fw_address***
IP address of the firewall that is between the source and destination hosts. A tunnel will be established between this host and the firewall. Therefore, the corresponding tunnel definition must be made on the firewall host. A host name can also be used for this flag and the first IP address returned by the name server for that host name will be used.
- g**
System auto-generated filter rule flag. If this flag is not used, the command will generate two filter rules for the tunnel automatically. The auto-generated filter rules will allow IP traffic between the two end points of the tunnel to go through the tunnel. If the **-g** flag is specified, the command will only create the tunnel definition and the user will have to add user-defined filter rules to let the tunnel work.
- h *src_ah_key***
Source AH Authentication Key. The input must be a hexadecimal string started with 0x. If this flag is not used, the system will generate a key using a random number generator.
- H *dst_ah_key***
Destination AH Authentication Key. The input must be a hexadecimal string started with 0x. If this flag is not used, the system will generate a key using a random number generator.
- k *src_esp_key***
Source ESP Encryption Key. This key is used by the source to create the tunnel. The input must be a hexadecimal string started with 0x. If this flag is not used, the system will generate a key using a random number generator.

-K *dst_esp_key*

Destination ESP Encryption Key. The input must be a hexadecimal string started with 0x. If this flag is not used, the system will generate a key using a random number generator.

-l *lifetime*

Key Lifetime, specified in minutes. This value indicates the time of operability before the tunnel expires. Upon expiration, the tunnel is removed from operation and from the tunnel cache maintained by the driver. The tunnel remains in the tunnel database upon expiration and is reactivated at the next IPL or by manual activation using the **acttun** command. The valid values are 0 - 44640. Value 0 indicates that the tunnel will never expire. The default value is 480.

-m *tunneltransport*

Secure Packet Mode. This value must be specified as **tunnel** or **transport**. The default value is **tunnel**. Tunnel mode will encapsulate the entire IP packet; transport mode only encapsulates the data portion of the IP packet. When generating a host-firewall-host tunnel (for a host behind a firewall), the value of **tunnel** must be used for this flag. The **-m** flag is forced to use the default value (**tunnel**) if the **-f** flag is specified.

-n *src_esp_spi*

Security Parameter Index for source ESP. This is a numeric value that, along with the destination IP address, identifies which security association to use for packets using ESP. Valid values for SPI are 256-4294967295. If this flag is not used, the system will generate an SPI for you.

-N *dst_esp_spi*

Security Parameter Index for the destination ESP. It must be entered if the policy specified in the **-P** flag includes ESP.

-p *src_policy*

Source policy. This flag identifies how the IP packet AH authentication or ESP authentication/encryption or both are to be used by this host. If specified as **ea**, ESP processing occurs before AH authentication. If specified as **ae**, ESP processing occurs after AH authentication. Specifying **e** alone or **a** alone corresponds to the IP packet being ESP authenticated/encrypted only or AH authenticated only. The default value for this flag will depend upon whether the **-e** and **-a** flags are supplied. The default policy will be **ea** if either both or neither the **-e** and **-a** flags are supplied; otherwise, the policy will reflect which of the **-e** and **-a** flags were supplied.

-P *dst_policy*

Destination policy, identifies how the IP packet AH authentication or ESP authentication/encryption or both are to be used by the destination. If specified as **ea**, ESP processing occurs before authentication. If specified as **ae**, ESP processing occurs after AH authentication. Specifying **e** or **a** corresponds to the IP packet being ESP authenticated or encrypted only or AH authenticated only. The default policy will be **ea** if either both or neither the **-E** and **-A** flags are supplied; otherwise, the policy will reflect which of the **-E** and **-A** flags were specified.

-s *src_host_IP_address*

Source Host IP address. This flag specifies the IP address of the local host interface to be used by the tunnel. A host name is also valid and the first IP address returned by name server for the host name will be used.

-u *src_ah_spi*

Security Parameter Index for source AH. Use SPI and the destination IP address to determine which security association to use for AH. If this flag is not used, the value of the **-n** SPI will be used.

-U *dst_ah_spi*

Security Parameter Index for the destination AH. If this flag is not used, the **-N** SPI will be used.

-x *dst_mask*

Network mask for the secure network behind a firewall. The Destination host is a member of the

secure network. The combination of **-d** and **-x** allows the source host to communicate with multiple hosts in the secure network through the source-firewall tunnel, which must be in tunnel mode. This flag is valid only when the **-f** flag is used.

-y [YIN]

Replay prevention flag. The valid values for the **-y** flag are **Y** (yes) and **N** (no). All encapsulations that are used in this tunnel (AH, ESP, sending, and receiving) will use the replay field if the value of this flag is **Y**. The default value is **N**.

Table 9. Supported authentication and encryption algorithms

Algorithm	Key Length (bytes)	AH	ESP Authentication	ESP Encryption
HMAC_MD5	16	X	X	
HMAC_SHA	20	X	X	
3DES_CBC	24			X
DES_CBC_8	8			X
DES_CBC_4	8			X
NULL	-			X

acttun command (adxipsat.386)

This section describes the **acttun** command.

Purpose

Activates tunnels.

Syntax

acttun [-t *tid_listall*] [-g] [-i]

Description

Use the **acttun** command to activate tunnels. This command will add the tunnel definitions to the tunnel cache maintained by the IPsec tunnel driver, thus, making the tunnel operational. If the **-g** flag is not specified, auto-generated and user-defined filter rules associated with this tunnel ID are marked active and the filter table is activated with the new tunnel filter definitions along with any previously activated filter rules.

Flags

- g** System auto-activate filter rule flag. If this flag is not used, the command will activate the auto-generated filter and user-defined filter rules for the tunnel automatically. The auto-generated filter rules will allow IP traffic between the two end points of the tunnel to go through the tunnel. If the **-g** flag is specified, the command will only activate the tunnels as specified by the flags and the user will have to manually activate filter rules to let the tunnel work.
- i** Initialization flag. If the **-i** flag is not used, all the tunnels in the tunnel database or those listed with the **-t** flag, if it is used, will be activated. If the **-i** flag is used, only the tunnels whose tunnel definitions in the tunnel database with the status of “active” will be activated. The **-t** flag is ignored when specified with **-i**. The **acttun -i** command is issued during IPL to activate the previous tunnel configuration.

-t *tid_listall*

If the **-t** flag is specified, only the tunnels that follows this flag will be activated. If the **-t** flag is not used or **-t all** is specified, all tunnels currently defined in the tunnel database will be activated. The *tid_list* can be a single tunnel ID or a sequence of tunnel IDs separated by a space, a comma (,) or a dash (-), where “-” specifies a range of IDs. For example, **-t 1,3,5-7** specifies a list of five tunnel IDs to be activated as follows: 1, 3, 5, 6 and 7. The maximum number of tunnel IDs that can be specified in the *tid_list* using any combination of formats is 500.

chtun command (adxipsct.386)

This section describes the **chtun** command.

Purpose

Changes a tunnel definition in the tunnel database.

Syntax

```
chtun -t tunnel_ID [-s src_host_IP_address] [-d dst_host_IP_address] [-m tunneltransport] [-f fw_address [-x dst_mask]] [-e src_esp_algo] [-a src_ah_algo] [-p src_policy] [-E dst_esp_algo] [-A dst_ah_algo] [-P dst_policy] [-l lifetime] [-k src_esp_key] [-h src_ah_key] [-K dst_esp_key] [-H dst_ah_key] [-n src_esp_spi] [-u src_ah_spi] [-N dst_esp_spi] [-U dst_ah_spi] [-b src_esp_auth_algo] [-c src_esp_auth_key] [-B dst_esp_auth_algo] [-C dst_esp_auth_key] [-y YIN]
```

Description

Use the **chtun** command to change a definition of a tunnel between a local host and a tunnel partner host. The tunnel must be inactive before it can be changed with the **chtun** command. Use the **rmtun** command to deactivate the tunnel first if it is currently active. One or more options can be specified to change in the tunnel definition. All other values remain as specified by the **gentun** command for that field. The **chtun** command can also change the auto-generated filter rules created for the tunnel by the **gentun** command.

Flags

-a *src_ah_algo*

Authentication algorithm, used by source host for IP packet authentication. The list of all authentication algorithms is shown Table 9 on page 65.

-A *dst_ah_algo*

Authentication algorithm, which is used by the destination for IP packet encryption. The list of all the authentication algorithms is shown in Table 9 on page 65.

-b *src_esp_auth_algo*

Authentication algorithm, used by source host for IP packet authentication. The list of all authentication algorithms is shown in Table 9 on page 65.

-B *dst_esp_auth_algo*

Destination ESP Authentication Algorithm. The list of all the authentication algorithms is shown in Table 9 on page 65.

-c *src_esp_auth_key*

Source ESP Authentication Key. It must be a hexadecimal string started with 0x.

-C *dst_esp_auth_key*

Destination ESP Authentication Key. It must be a hexadecimal string started with 0x.

-d *dst_host_IP_address*

Destination Host IP address. For a host-host tunnel, this value is the IP address of the destination host interface to be used by the tunnel. For a host-firewall-host tunnel, this is the IP address of a destination host behind the firewall. A host name is also valid and the first IP address returned by the name server for the host name will be used.

-e *src_esp_algo*

Encryption algorithm, used by source host for IP packet encryption. The list of all encryption algorithms is shown in Table 9 on page 65.

-E *dst_esp_algo*

Encryption algorithm, which is used by the destination for IP packet encryption. The list of all the encryption algorithms is shown in Table 9 on page 65.

-f *fw_address*

IP address of the firewall that is between the source and destination hosts. A tunnel will be established between the source and the firewall; therefore, the corresponding tunnel definition

must be made in the firewall host. A host name can also be specified with this flag, and the first IP address returned by name server for the host name will be used. The **-m** flag is forced to use default value (**tunnel**) if **-f** is specified.

-h *src_ah_key*

The Key String for source AH. The input must be a hexadecimal string started with 0x.

-H *dst_ah_key*

The Key String for destination AH. The input must be a hexadecimal string started with 0x.

-k *src_esp_key*

The Key String for the source ESP used by the source to create the tunnel. The input must be a hexadecimal string started with 0x.

-K *dst_esp_key*

The Key String for destination ESP. The input must be a hexadecimal string started with 0x.

-l *lifetime*

Lifetime, specified in minutes. The value of this flag indicates the time of operability before the tunnel expires. Upon expiration, the tunnel is removed from operation and from the tunnel cache maintained by the driver. The tunnel remains in the tunnel database upon expiration and is reactivated at the next IPL or by manual activation using the **acttun** command. The valid values are 0-44640. Value 0 indicates that the tunnel will never expire.

-m *tunneltransport*

Secure Packet Mode. This value must be specified as **tunnel** or **transport**.

-n *src_esp_spi*

Security Parameter Index for source ESP. This SPI and the destination IP address are used to determine which security association to use for ESP.

-N *dst_esp_spi*

Security Parameter Index for the destination ESP.

-p *src_policy*

Source policy, identifies how the IP packet authentication, encryption, or both are to be used by the source. If the value of this flag is specified as **ea**, the IP packet gets encrypted before authentication. If specified as **ae**, it gets encrypted after authentication. Specifying **e** or **a** alone corresponds to the IP packet being encrypted only or authenticated only.

-P *dst_policy*

Destination policy, identifies how the IP packet authentication, or encryption, or both are to be used by the destination. If the value of this flag is specified as **ea**, the IP packet gets encrypted before authentication. If specified as **ae**, it gets encrypted after authentication, whereas specifying **e** or **a** alone corresponds to the IP packet being encrypted only or authenticated only.

-s *src_host_IP_address*

Source Host IP address. This flag specifies the IP address of the local host interface to be used by the tunnel. A host name is also valid and the first IP address returned by name server for the host name will be used.

-t *tunnel_ID*

The tunnel identifier (ID) is a locally unique, numeric identifier for a particular tunnel definition. The value must match an existing tunnel ID.

-u *src_ah_spi*

Security Parameter Index for source AH. This SPI and the destination IP address is used to determine which security association to use for AH.

-U *dst_ah_spi*

Security Parameter Index for the destination AH.

-x *dst_mask*

This flag is used for host-firewall-host tunnels. The value is the network mask for the secure network behind a firewall. The Destination host specified with the **-d** flag is a member of the secure network. The combination of the **-d** and **-x** flags allows source host communications with multiple hosts in the secure network through the source-firewall tunnel, which must be in tunnel mode. This flag is valid only when **-f** is specified.

-y YIN Replay prevention flag. The valid values for the **-y** flag are **Y** (yes) and **N** (no).

rmtun command (adxipsrt.386)

This section describes the **rmtun** command.

Purpose

Deactivates an operational tunnel or tunnels and optionally removes tunnel definitions from the tunnel configuration database.

Syntax

rmtun -t *tid_list* | all [-d]

Description

Use the **rmtun** command to deactivate an active tunnel or tunnels by removing the tunnels from the tunnel cache maintained by the tunnel driver. Specify **-d** to optionally remove tunnel definitions from the tunnel database. It also removes the auto-generated filter rules created for the tunnel by the **gentun** command when the tunnel definition is removed from the tunnel database. User-defined filter rules that direct traffic through the tunnel being removed are changed to a negative tunnel ID. For example, deleting tunnel 5 changes a user-defined filter rule configured with tunnel ID 5 to -5. Note that the filter table is activated on any filter rule change and any previously active filters rules will be reactivated.

Flags

-d Specifies that the tunnels are to be removed from the tunnel database. This is an optional flag.

-t *tid_list* | all

The **-t** flag is used to specify which tunnels to deactivate. The *tid_list* variable specifies the list of tunnel IDs to deactivate. Tunnel IDs are separated by a space, comma (,), or dash (-), where “-” specifies a range of IDs. For example, **-t 1,3,5-7** specifies a list of five tunnel IDs to be removed as follows: 1, 3, 5, 6 and 7. If **all** is specified for the **-t** flag, all tunnels are deactivated. If the **-d** flag is also specified, all the tunnel definitions in the list are removed from the tunnel database.

Istun command (adxipslt.386)

This section describes the **Istun** command.

Purpose

Lists tunnel definitions.

Syntax

Istun [-t *tid_list*] [-a]

Description

Use the **Istun** command to list the tunnel definitions and their current status. This command can either list the tunnels in the tunnel configuration database or the tunnels that are active in the driver.

Flags

-a Lists the tunnels active in the driver’s tunnel cache.

-t *tid_list*

Only list the tunnel definition and current status for the tunnels whose tunnel IDs are in *tid_list*. If this flag is not used, all the tunnel definitions and their current status will be listed.

ipsecstat command (adxipsst.386)

This section describes the **ipsecstat** command.

Purpose

Lists status of IP Security devices and statistics for IP Security packets.

Syntax

ipsecstat [-c] [-d]

Description

The **ipsecstat** command, used without flags, displays the status of the IP Security drivers and the statistics of IP Security packets. The command can be used with flags to only list the status of IP Security drivers or to reset statistics counters to zero.

Flags

-c Resets statistics counters (after displaying current value). The **-c** flag cannot be used with the **-d** flag.

-d Lists only the status of the IP Security drivers. The **-d** flag cannot be used with the **-c** flag.

Sample output

The following example shows a sample of output from the command: **ipsecstat**:

IPsec Support: Available

```
IPSec Statistics -
Total incoming packets:          454
  Incoming AH packets:          100
  Incoming ESP packets:          100
Total outgoing packets:          116
  Outgoing AH packets:          100
  Outgoing ESP packets:          100
Total incoming packets dropped:    0
  Filter denies on input:         0
  AH did not compute:             0
  ESP did not compute:            0
  AH replay violation:            0
  ESP replay violation:           0
  IPsec tunnel/policy mismatch:    0
Total outgoing packets dropped:    0
  Filter denies on output:         0
Tunnel cache entries added:        2
Tunnel cache entries expired:      0
Tunnel cache entries deleted:      0
```

Chapter 4. Using Secure Shell in the Operating System

This chapter describes using Secure Shell (SSH) in your system.

Notes:

1. The options included in this product can help your company address the PCI DSS requirements.
2. The customer is responsible for evaluation, selection and implementation of security features, administrative procedures and appropriate controls.
3. PCI DSS is Payment Card Industry Data Security Standards.

Introduction to Secure Shell

Secure Shell (SSH) is a standards-based application and an associated set of networking protocols that provides secure, encrypted communications over an untrusted network (for example, the Internet) between servers and clients. It allows password validation, communications encryption, compression, and tunneling. RFC 4251 is available on the Internet and provides a detailed description of SSH. This section describes how SSH is implemented for 4690 OS V6.

The version of SSH that the 4690 OS V6 uses is known as SSH-2, referred to as SSH in this and other 4690 OS publications. The SSH software consists of a client (SSH) and a server (SSHD).

The 4690 OS SSH-2 implementation provides a secure interactive shell session and secure FTP (SFTP) service.

No other SSH-2 functions (such as REXEC) are supported.

The interactive shell session(s) uses the current 4690 OS Enhanced Telnet service along with the SSH-2 authentication protocol and secure data encrypted channel. The 4690 OS implementation of the SSH-2 suite supports up to a total of eight concurrent interactive sessions. That is, eight sessions of all SFTP connections, eight Secure Telnet sessions, or a mix of SFTP and Secure Telnet sessions totaling eight.

It is important to note that the 4690 OS TCP/IP stack only supports a maximum of eight outstanding requests which also limits the number of SFTP sessions. In order to operate within these constraints, it is important to assign only one outstanding request per SFTP session (-R option). However, it is possible to assign more outstanding requests per session if there are less active concurrent sessions. The 4690 OS TCP/IP stack can reliably handle communication buffers (-B option) up to 16384 bytes, so the optimum buffer size is 4096 bytes per SFTP session.

Note: Refer to the *4690 OS: Planning, Installation, and Configuration Guide* for information about installing SSH and configuring the system to start SSH automatically during store controller IPL.

SSH commands

The SSH commands are described in the following sections.

Secure Shell Server (ADXSSHD.L386)

The SSH server (ADXSSHD.L386) must be running in order to allow the Secure Shell clients to connect.

The server can be started from a command line session or as a background application.

The SSH server listens for connections from SSH clients residing on hosts. A connection can be established through the command line interface or as a background application. Multiple instances of the server can be started to handle incoming host connections. The server handles key exchange, encryption, authentication, and data exchange.

Each host has a host-specific key (RSA or DSA) used to identify the host. A key agreement is used when a session starts to provide a shared session key.

The rest of the session is encrypted using a symmetric cipher. The client selects the encryption algorithm to use from those offered by the server. SSH-2 uses a public-key-based user password authentication and challenge-response based methods.

If the client successfully authenticates itself, a dialog for preparing the session is entered. Finally, the client requests a shell. The client and server then enter session mode. In this mode either side can send data at any time and such data is forwarded to or from the shell on the server side and the user terminal on the client side. When the user terminates the session, the server sends command status to the client and both sides exit.

The SSH server can be configured using command line options or a configuration file (by default `c:\adx_sdt1\ADXSSHDF.DAT`). Command line options override values specified in the configuration file.

Syntax

ADXSSHD.L386 [-d] [-f *config_file*] [-h *host_key_file*] [-p *port*]

Command line options

-d Debug. Multiple **-d** options (up to 3) increase the debugging detail. When in debug mode, the server can have only one client session. The log file will be placed into `c:\adx_sdt1`.

The **-d** option should be used only when directed by support personnel.

-f *config_file*

Overrides the default configuration file, `c:\adx_sdt1\adxsshd.f.dat`. The SSH server will not start if the configuration file cannot be found.

-h *host_key_file*

Specifies a file from which a host key is read. The defaults are `adx_sdt1:ADXSSHRK.DAT` and `adx_sdt1:ADXSSHDK.DAT` for RSA and DSA private host key files. It is possible to have multiple host key files for different host key algorithms.

-p *port* Specifies the port on which the server listens for connections. The default port is 22. Multiple port options are permitted. Ports specified in the configuration file are ignored when a command line port is specified.

Key files

You can import key files from UNIX-like machine provided that the key files are OpenSSH implementation compatible; otherwise, the key files will not be readable by the SSH-2 4690 OS suite.

Authorized client key files

By default the SSH server searches `c:\adx_sdt1\ADXSSHAK.DAT` and `c:\adx_sdt1\ADXSSHBK.DAT` files for the RSA and DSA public portion of the client keys. If neither of the files are present, the server will prompt for the user password if the configuration file allows password authentication.

Host key files

At start time, the SSH server will check for the existence of the host key files, `c:\adx_sdt1\ADXSSHDK.DAT` and `c:\adx_sdt1\ADXSSHRK.DAT`. If these files do not exist, the SSH server will generate a new set. This set will include the aforementioned files plus their corresponding public key files: `c:\adx_sdt1\ADXSSHDK.PBK` and `c:\adx_sdt1\ADXSSHRK.PBK`. These public files should be added to any client that will communicate with the server.

Log file

By default, the SSH Server log traces error level and fatal level log entries to a file named `ADX_SDT1:SSHLOG.LOG`. When the log file reaches 3MB in size, the log is flushed to a history file named `ADX_SDT1:SSHLOG.001`.

The server and its child processes trace to the same file, so the process ID is appended to each entry. Following is a sample log entry: ID:d6 2010/03/22 10:12:10.439 fatal: Cannot bind any address.

The **-q** command line option will allow the user to deactivate the default SSH logging. This is not recommended as it will eliminate necessary debug tracing used by 4690 OS support.

This default level of SSH logging is overridden by either:

- The LogLevel property setting in the SSH server configuration file, ADX_SDT1:ADXSSHDF.DAT, or
- The use of the command line argument **-d**

When detailed debug logging is activated by either of these methods, a log file, ADX_SDT1:SSHxxxxx.LOG is generated, where xxxxx is the server process id.

Non-4690 OS clients

The SSH server allows incoming connections from non-4690 OS clients, such as UNIX or Linux machines, but some considerations apply:

1. SFTP clients should use the **-B** option or equivalent to set a buffer of a maximum size of 16,384 bytes.
2. SFTP clients use the **-R** option to set a maximum of 8 outstanding requests at one time.

4690 OS account lockout

For 4690 OS, it is possible to create an account lockout policy that locks a user account and discontinues the session after some predetermined number of unsuccessful login attempts. This lockup applies to both SFTP (ADXSSHFL.386) and SSH (ADXSSHCL.386) users.

The SSH server uses lockout settings that you can access by choosing: **System Configuration -> System Security -> FTP ID Security**.

Note: The 4690 SSH server uses the lockout settings under **FTP ID Security** that are independent from 4690 FTP server; therefore, a user locked out by SSH server is not locked out of FTP login nor locked out of console login.

The lockout settings are used by the SSH server for user lockout regardless of whether or not that user wants a Secure FTP session or Secure Telnet session.

Troubleshooting the SSH Server

A W650 message will be generated if the SSH Server fails to start. To aid in problem determination, the RC value in the W650 message may indicate the problem:

- | | |
|-----------|---|
| 2 | Enhanced password has not been enabled |
| 3 | Too many ports specified |
| 4 | Bad port number specified |
| 5 | Too many host keys |
| 6 | Extra arguments were specified |
| 7 | No hostkeys available |
| 8 | Bad server key size |
| 9 | Error reading server configuration file |
| 10 | Usage error (bad parameters) |

Secure Shell Client (ADXSSHCL.386)

ADXSSHCL.386 is the 4690 OS SSH-2 client. The SSH client supports password and public key authentication.

Syntax

ADXSSHCL.386 [-CVv] [-c *cipher_spec*] [-F *config_file*] [-i *identity_file*] [-l *login_name*] [-o *option*] [-p *port*]
[[*user@host*] [*command*]

Command line options

- C** Requests data compression for stdin and stdout.
- V** Prints the version number and exits.
- v** Sets the verbose mode to active. A log file is generated. The detail can be increased by adding up to 3 **-v** options.

-c *cipher_spec*

Specifies a cipher or list of ciphers. Ciphers in a list should be in order of preference and separated by commas. The default cipher list is: aes128-cbc, 3des-cbc, aes192-cbc, aes256-cbc, aes128-ctr, aes246-ctr.

-F *config_file*

Sets the configuration file to be read. The default is c:\adx_sdt1\ADXSSHCF.DAT. This configuration file is not required to be present for the SSH client to operate. It should be created by the user if default options are to be changed.

-i *identity_file*

Sets a file to use for public key authentication. The file contains the private key for the user. The default private key files are c:\adx_sdt1\ADXSSHRC.DAT for RSA keys and c:\adx_sdt1\ADXSSHDC.DAT for DSA keys.

-l *login_name*

Sets the user name to use for logging in on the remote server.

-o *option=value*

Allows the user to directly set various options using the option format of the configuration file. Supported options are:

- Host
- Ciphers
- Compression
- ConnectionAttempts
- GlobalKnownHostsFile
- HostName
- IdentityFile
- LogLevel
- NumberOfPasswordPrompts
- PasswordAuthentication
- PubkeyAuthentication
- StrictHostKeyChecking
- TCPKeepAlive
- User
- UserKnownHostsFile

Refer to "SSH client configuration file (ADXSSHCF.DAT)" on page 83 for a description of the options.

- p** *port* Sets the port to connect to on the remote server. The default is 22.

Note: With SSH/SFTP the **-p** flag is by design, not supported when using the GET and PUT interactive commands. The "Preserve timestamp" option in WinSCP translates to the **-p** option on SFTP GET and PUT commands. According to the standard, if the **-p** flag is specified on a GET or PUT, then the full file permissions and access times are copied to the remote host. In SSH2 terms, the command is SETSTAT (9) with at attribute to modify the file timestamp (8).

If you change the WinSCP settings 'Preserve time stamp' to disable, then the SSH2 command will not be rejected by the 4690 SFTP server as an unsupported operation.

[user@]host

Indicates the host name or IP address to connect to. The user name given is the user name on the remote host.

command

Command to execute on remote host. The W992 system message will be logged to indicate that the command was started by the SSH server, the W620 system message will be logged by SSH server to indicate that the command has ended and will include the command return code. Upon completion of the command, the SSH server will send a package disconnect message to the client.

Note: The user should have authority for the following Enhanced Password attributes to run the requested command:

```
ADX_CPW_ATTR_SSH_SECURE_REMOTE_LOGON
ADX_CPW_ATTR_BACKGROUND_APPLICATION_CONTROL
ADX_CPW_ATTR_COMMAND_MODE
```

If the user does not have the proper authority to run the requested command, then the W979 system message will be logged if a batch file is used to run a command, it should include error handling with error level and exit, with a return code different to 0.

The following is an example of SSH client that would start a batch file to copy a file:

```
adxsshcl user@X.X.X.X path\mycopy.bat source destination
```

where XXX is your IP address, the name of the batch file should be qualified including path name, name of the batch file including extension name and followed by a list of parameters if needed.

mycopy.bat which receives two parameters to copy a file including error handling:

```
copy %1 %2
if errorlevel 0 goto exitok
if not errorlevel 0 goto error
:exitok
echo %1 was copied to %2
exit 1
:error
echo Error found copying %1 to %1
exit 2
```

if either .286 or .386 program is used to run a command, the name of the program should be qualified including: path name, name of the .286 or .386 file including extension name and followed by a list of parameters if needed.

The following is an example of SSH client that would run a .286 command to rename a file:

```
adxsshcl user@X.X.X.X c:\adx_spgm\rename.286 name1 name2"
```

Known Hosts file

The file c:\adx_sdt1\adxsshkh.dat contains the public keys for all hosts the user has logged into. The user can append host public keys.

Identify files

ADXSSSRC.DAT and ADXSSSHDC.DAT in c:\adx_sdt1 are the default placeholders for the user RSA and DSA private keys. If the user decides to use the default names for such files, the private keys should be protected by a passphrase because all users of the SSH client on that machine will share the default

private key placeholders. It is the responsibility of the administrator or the user to update the respective host server with the public key part generated for each private key in use. The user should append the files used for the private keys with any new private keys that need to be added.

Log file

When debug information is requested, a log file, c:\adx_sdt1\SSHxxxxx.LOG is generated, where xxxxx is the server process id.

Secure FTP server (ADXSSHPL.386)

ADXSSHPL.386 is the SFTP server for the SSH-2 suite in the 4690 OS. It is called from within the SSH server and does not have any command line options and is not intended to be invoked by the user. The following table gives the SFTP interactive commands that are supported by the 4690 OS. Refer to “Secure FTP console session (ADXSSHFL.386)” on page 77 for descriptions of the commands.

Table 10. SFTP interactive commands supported by the 4690 OS SFTP server

Command	From a UNIX-like machine to 4690 OS
bye	Supported
cd	Supported
chdir	Supported
chgrp	Not supported
chmod	Not supported
chown	Not supported
dir	Supported
df	Supported Note: The -hi flags are NOT supported by the server.
exit	Supported
get	Supported Note: The -P flag is NOT supported by the server.
mget	Supported
help	Up to local client
lcd	Up to local client
lchdr	Up to local client
lls	Up to local client
lmdir	Up to local client
ln	Not supported
lpwd	Up to local client
ls	Supported
lumask	Up to local client
mkdir	Supported
progress	Up to local client
put	Supported Note: The -P flag is NOT supported by the server.
mput	Supported
pwd	Supported
quit	Supported
rename	Supported

Table 10. SFTP interactive commands supported by the 4690 OS SFTP server (continued)

Command	From a UNIX-like machine to 4690 OS
rm	Supported
rmdir	Supported
symlink	Not supported
version	Up to local client
!	Up to local client
?	Up to local client

Multiple drive unit support

Because 4690 OS differs from UNIX in areas such as hierarchical file system and handling of drive units, the user must issue a **cd drive:** command, where *drive* is the drive letter, to set that drive as the working drive. After the user has selected a working drive, each subsequent path entered will be relative to it. When typing paths, always use the forward slash "/" as you would at any UNIX console.

Permission file

The SFTP server uses the file ADXSSHXH.DAT to set the different path permissions that a user is authorized to have for a SFTP session. ADXSSHXH.DAT resides in c:\adx_sdt1. This file can not be updated by means of an SFTP service.

Command line logging

Use Command line logging to log the commands issued to/from the Secure FTP Server. Refer to *Command line logging* in the 4690 OS User's Guide for more information.

Secure FTP console session (ADXSSHFL.386)

ADXSSHFL.386 is the 4690 OS Secure FTP (SFTP) console session equivalent from the SSH-2 suite. It requires the SSH client to communicate with an SSH server. Only an interactive session is supported at this time.

Syntax

SFTPIADXSSHFL.386 [-Cv] [-B *buffer_size*] [-b *batch_file*] [-F *config_file*]] [-o *config_option*] [-R *outstanding_requests*] *host*

Command line options

-C Requests data compression from the SSH client.

-v Sets the verbose mode in the SSH client to active.

-B *buffer_size*

Sets the buffer size that the SFTP console session will use when transferring data.

Note: Set the buffer size to 16384 bytes when this client is connected to a 4690 OS server.

-b *batch_file*

Specifies a batch file from which SFTP console session should read a series of commands instead of reading commands from the system console.

-F *config_file*

Sets a configuration file for the SSH client.

-o *config_option=value*

Specifies an option to pass directly to the SSH client using the SSH client configuration file format. Supported options are:

Host

Ciphers

Compression
 ConnectionAttempts
 GlobalKnownHostFiles
 HostName
 IdentityFile
 LogLevel
 NumberOfPasswordPrompts
 PasswordAuthentication
 PubkeyAuthentication
 StrictHostKeyChecking
 TCPKeepAlive
 User
 UserKnownHostsFile

See “SSH client configuration file (ADXSSHCF.DAT)” on page 83 for a description of the options.

-R *outstanding_requests*

Sets how many requests can be outstanding at a given time. When establishing a connection to a 4690 OS server, use a maximum of 8 outstanding requests.

host Indicates the host name or IP address to connect to.

Supported SFTP interactive commands

Table 11 gives the interactive commands that are supported by ADXSSHFL.386.

Table 11. SFTP interactive commands supported by ADXSSHFL.386

Command	Description	From 4690 OS to 4690 OS	From 4690 OS to UNIX
bye	Ends ADXSSHFL.386.	Supported	Supported
cd <i>path</i>	Same as the chdir <i>path</i> command. Changes the remote directory to <i>path</i> . Also at the 4690 OS server, it changes the working unit.	Supported	Supported
chdir <i>path</i>	Same as the cd <i>path</i> command. Changes the remote directory to <i>path</i> . Also at the 4690 OS server, it changes the working unit.	Supported	Supported
chgrp <i>grp path</i>	Changes the group of the file, <i>path</i> , to <i>grp</i> .	Not supported	Supported
chmod <i>mode path</i>	Changes the file permissions to <i>mode</i> .	Not supported	Supported
chown <i>own path</i>	Changes the file owner to <i>own</i> .	Not supported	Supported
dir [<i>flags</i>] [<i>path</i>]	Same as ls command. Displays the remote working directory of remote path. Flags: 1- Produce single column output. l - Display additional details, including permissions and ownership information.	Supported	Supported
df	Displays statistics for filesystem containing path	Supported	Supported
exit	Ends ADXSSHFL.386	Supported	Supported
get <i>remote_path</i> [<i>local_path</i>]	Same as the mget <i>remote_path</i> [<i>local_path</i>] command. Requests <i>remote_path</i> and saves it on the local machine renaming it to <i>local path</i> , if specified.	Supported	Supported
help	Displays help text.	Supported	Supported
lcd <i>path</i>	Same as the lchdir <i>path</i> command. Changes the local directory to <i>path</i> .	Supported	Supported

Table 11. SFTP interactive commands supported by ADXSSHFL.386 (continued)

Command	Description	From 4690 OS to 4690 OS	From 4690 OS to UNIX
lcd <i>path</i>	Same as the lcd <i>path</i> command. Changes the local directory to <i>path</i> .	Supported	Supported
lls [[<i>options</i>] [<i>path</i>]]	Displays the local directory at <i>path</i> or the current directory, if none is specified. <i>options</i> can be any of the 4690 OS dir command options.	Supported	Supported
mkdir <i>pathpath</i>	Create local directory to <i>path</i> .	Supported	Supported
ln <i>oldpath newpath</i>	Creates a <i>newpath</i> symbolic link from <i>oldpath</i> .	Not supported	Supported
lpwd	Displays current local working directory.	Supported	Supported
ls [<i>flags</i>] [<i>path</i>]	Same as the dir [<i>flags</i>] [<i>path</i>] command. Displays the remote working directory of remote path. Flags: 1- Produce single column output. l - Display additional details, including permissions and ownership information.	Supported	Supported
lumask	Set local unmask to unmask .	Not supported	Not supported
mget <i>remote_path</i> [<i>local_path</i>]	Same as the get <i>remote_path</i> [<i>local_path</i>] command. Requests <i>remote_path</i> and saves it on the local machine renaming it to <i>local_path</i> , if specified.	Supported	Supported
mput <i>local_path</i> [<i>remote_path</i>]	Same as the put <i>local_path</i> [<i>remote_path</i>] command. Saves the <i>local_path</i> to the <i>remote_path</i> or to the remote working directory.	Supported	Supported
mkdir <i>path</i>	Creates remote <i>path</i> directory.	Supported	Supported
progress	Toggles the display of the progress meter.	Supported	Supported
put <i>local_path</i> [<i>remote_path</i>]	Same as the mput <i>local_path</i> [<i>remote_path</i>] command. Saves the <i>local_path</i> to the <i>remote_path</i> or to the remote working directory.	Supported	Supported
pwd	Display the remote working directory.	Supported	Supported
quit	Quit ADXSSHFL.386.	Supported	Supported
rename <i>oldpath newpath</i>	Renames the <i>oldpath</i> to <i>newpath</i> .	Supported	Supported
rm <i>path</i>	Erases the remote <i>path</i> .	Supported	Supported
rmdir <i>path</i>	Erases the remote directory <i>path</i> .	Supported	Supported
symlink <i>oldpath newpath</i>	Creates a symbolic link from <i>oldpath</i> to <i>newpath</i> .	Not supported	Supported
version	Displays the SFTP version information.	Supported	Supported
! <i>command</i>	Executes <i>command</i> in the local shell.	Supported	Supported
?	Displays help text.	Supported	Supported

Command line logging

Use Command line logging to log the commands issued to/from the Secure FTP Client. Refer to *Command line logging* in the 4690 OS User's Guide for more information.

SSH key generator (ADXSSHL.386)

The SSH key generator (ADXSSHL.386) creates, manages, and converts authentications keys to be used by a SSH-2 suite. It can generate RSA and DSA keys.

Generated keys can be protected by a passphrase, which is a password for the key itself. Passphrases should be between 10 to 30 characters long, including upper and lower case and numeric characters.

Note: It is important to remember passphrases. Passphrases can not be recovered, they can only be changed.

For each generated key, two files are created. One file contains the private part of the key and the second file contains the public part of the key. If no key file is specified, the default files created are c:\adx_sdt1\adxsshrc.dat and c:\adx_sdt1\adxsshrc.pbk for RSA keys or c:\adx_sdt1\adxsshdc.dat and c:\adx_sdt1\adxsshdc.pbk for DSA keys.

Syntax

ADXSSHL.386 [-q] [-b *bits*] -t *type* [-N *new_passphrase*] [-C *comment*] [-f *keyfile*]

ADXSSHL.386 -p [-P *old_passphrase*] [-N *new_passphrase*] [-f *keyfile*]

ADXSSHL.386 -y [-f *keyfile*]

ADXSSHL.386 -l [-f *keyfile*]

Command line options

-q Suppresses messages and prompts from the tool.

-b *bits* Specifies the number of bits in the key to be created. The default is 1024.

DSA accepts 1024 bits only. RSA accepts 768 - 16384 range.

-t *type* Specifies whether a RSA or DSA key should be created. Possible values are *rsa* or *dsa*.

-C *comment*

Specifies a comment. Use the comment to identify keys. Comments should be enclosed in a pair of double quotes (" ").

-f *keyfile*

Sets a key file name. The private part of the key will be saved with the name given on parameter *keyfile* and the public key file will be created in a file of the same name, but with the extension .pbk. For this reason the extension .pbk should not be used in *keyfile* parameter. When you edit a key file, type the file name and its extension. The typed name should be the name of the private key file.

Note: Keyfiles created using this naming option will be copied to target controllers by the Lan Disk Rebuild Utility.

-p Changes the passphrase of a private key file. If a key file was not specified (**-f *keyfile***), the name will be requested.

-P *old_passphrase*

Specifies the old passphrase to be changed.

-N *new_passphrase*

Specifies the new passphrase.

Note: Host keys must have an empty passphrase, so this option must not be used when generating a host key.

-y Reads a private key file and prints an OpenSSH public key.

-l Displays the fingerprint of a public key file.

Key files

The default path for key files is c:\adx_sdt1. In general, the private keys should be added (appended) to the ADXSSSHRC.DAT or ADXSSHDC.DAT files on the 4690 OS client machine and their public counterparts, ADXSSSHRC.PBK and ADXSSHDC.PBK should be added to ADXSSHAK.DAT or ADXSSHBK.DAT or both on the 4690 OS server machine. Key files generated by the SSH key generator can usually be exported to UNIX- like machines, provided that they support the OpenSSH key format.

Configuration files

This section describes the configuration files associated with SSH and their formats. These files are located in the c:\adx_sdt1 directory.

SSH server configuration file (ADXSSHDF.DAT)

ADXSSHDF.DAT is the 4690 OS server configuration file for SSH support. If this file is not present, the SSH server will not start. If the file is present, but empty, the SSH server will start.

Sample file

```
# OS 4690's ADXSSHD.L386 server configuration file for SSH support
# Ver. 2.
# See the readme file for more information.

# Here you will find a list of the supported options for the
# ADXSSHD.L386's config file. Like its counterpart, the OpenSSH, the
# options will appear with its default values where possible and you
# can leave them commented.
# Uncommenting options will change a default value.

#AllowUsers
#AuthorizedKeysFile      c:\adx_sdt1\adxsshak.dat

#banner

#Ciphers aes128-cbc,3des-cbc,aes192-cbc,aes256-cbc,aes128-ctr,
#      aes192-ctr,aes256-ctr
#Compression yes

#DenyUsers

#HostKeys for SSH protocol version 2
# rsa file
#HostKey c:\adx_sdt1\adxsshrk.dat
# dsa file
#HostKey c:\adx_sdt1\adxsshdh.dat

#PasswordAuthentication yes
#PubkeyAuthentication yes
#Port 22

# WARNING: Never change this option, it is vital for SFTP
# support. No other values are supported.
Subsystem      sftp      adx_spgm:adxsshpl.386

#TCPKeepAlive yes
```

Where:

- # Marks a comment. After this symbol, the rest of the line will be ignored. You can uncomment the desired option by removing the # symbol and changing the defaults of the option.

AllowUsers

The default is that the login is permitted for all users. You can enter a list of users to restrict the access to the users in the list. Separate the user names by spaces.

Authorized KeysFile

Sets the path and file containing the client public keys that will be used for user authentications. The defaults are c:\adx_sdt1\ADXSSHAK.DAT and c:\adx_sdt1\ADXSSHBK.DAT for RSA and DSA.

Banner

In some jurisdictions, giving a warning message before authentication might be necessary for legal protection. The contents of the specified file are sent to the remote user before authentication is performed. By default, no banner is displayed.

Note: If key authentication is selected and the key does not have a passphrase, the banner is displayed too briefly to be noticed. This behavior can be corrected by adding the warning message to a user-configured logon screen (ADXLOGOD.DAT), and it is seen when the secure shell session starts.

Ciphers

Ciphers are allowed. The defaults are aes128-cbc, 3des-cbc, aes192-cbc, aes256-cbc, aes128-ctr, aes192-ctr, and aes256-ctr.

ClientAliveCountMax

Sets the number of client alive messages which may be sent without the SSH Server receiving any messages back from the client. If this threshold is reached while client alive messages are being sent, the SSH Server will disconnect the client, terminating the session. It is important to note that the use of client alive messages is very different from TCPKeepAlive. The client alive messages are sent through the encrypted channel and therefore will not be spoofable. The TCP keepalive option enabled by TCPKeepAlive is spoofable. The client alive mechanism is valuable when the client or server depend on knowing when a connection has become inactive. The default value is **3**. If ClientAliveInterval is set to 15, and ClientAliveCountMax is left at the default, unresponsive ssh clients will be disconnected after approximately 45 seconds.

ClientAliveInterval

Sets a timeout interval in seconds after which if no data has been received from the client, the SSH Server will send a message through the encrypted channel to request a response from the client. The default is 0, indicating that these messages will not be sent to the client. This option applies to protocol version 2 only.

Note: The recommended value is 5.

Compression

Set to "yes" to use data compression.

DenyUsers

List of users to not allow access. User names should be separated by spaces.

HostKey

Sets the file that contains a private host key for the SSH server. The defaults are: c:\adx_sdt1\adxsshrk.dat and c:\adx_sdt1\adxsshdh.dat.

PasswordAuthentication

Enables or disables password authentication. The default is "yes".

PubKeyAuthentication

Enables or disables public key authentication. The default is "yes".

Port Sets the port number that the SSH server listens on. The default is 22.

Subsystem

Sets an external subsystem.

TCPKeepAlive

The default is "yes". This option allows the system to send TCP keepalive messages so it can detect when a connection has ended and close sessions with such a client.

SSH client configuration file (ADXSSHCF.DAT)

ADXSSHCF.DAT is the 4690 OS SSH-2 client configuration file. This file is not required for the SSH client (ADXSSHCL.386) to start and operate.

It is the responsibility of the user to create and maintain ADXSSHCF.DAT if it needed to override the SSH client defaults.

Sample file

```
# OS 4690's ADXSSHCL.386 client configuration file for SSH support Ver. 2.
# See the readme file for more information.

# Here you will find a list of the supported options for the ADXSSHD.386's
# config file. Like its counterpart, the OpenSSH, the options will appear
# with its default values where possible and you can leave them commented.
# Uncommenting options will change a default value.

# Configuration values are changed the first time they appear, therefore
# Host specific values should be at the beginning of the configuration
# and any defaults at the end.

# This file can have several 'Host' entries with its own options in effect
# until the next appearance of the next 'Host' option.

# Host *
#   Ciphers aes128-cbc,3des-cbc,aes192-cbc,aes256-cbc
#   Compression no
#   ConnectionAttempts 1
#   GlobalKnownHostsFile c:\adx_sdt1\adxssshk.dat
#   HostName
#   IdentityFile c:\adx_sdt1\adxssshrc.dat c:\adx_sdt1\adxssshdc.dat
#   LogLevel INFO
#   NumberOfPasswordPrompts 3
#   PasswordAuthentication yes
#   Port 22
#   PubkeyAuthentication yes
#   StrictHostKeyChecking ask
#   TCPKeepAlive yes
#   User
#   UserKnownHostsFile c:\adx_sdt1\adxssshk.dat
```

Where:

Marks a comment. After this symbol, the rest of the line will be ignored. You can uncomment the desired option by removing the # symbol and changing the defaults of the option.

Host Make the following options apply to the specific host; otherwise, options will apply to the host name passed as an argument to the SSH client (ADXSSHCL.386).

Ciphers

Sets the cipher to be used for encryption or a list of ciphers, in order of preference, separated by commas. The default is aes128-cbc,3des-cbc,aes192-cbc,aes256-cbc.

Compression

Set to "yes" to use data compression. The default is "no".

ConnectionAttempts

Specifies the number of tries per second to make before exiting in case the connection fails.

GlobalKnownHostsFile

Sets the host key file to use instead of c:\adx_sdt1\adxssshk.dat.

HostName

Specifies the real host name to login to.

IdentityFile

Sets the file name for the user private RSA or DSA files. These names will override the default names and paths. The default names and paths are c:\adx_sdt1\adxssshrc.data and c:\sdt1\adxssshdc.dat.

LogLevel

Specifies the detail to log. The default is INFO. Possible values are QUIET, FATAL, ERROR, INFO, VERBOSE, DEBUG1, DEBUG2, and DEBUG3.

NumberOfPasswordPrompts

Sets the number of password prompts. The default is 3. The maximum is 6.

PasswordAuthentication

Specifies whether or not password authentication can be used. The default is "yes".

Port Sets the port number to connect to on the server. The default is 22.

PubkeyAuthentication

Specifies if public key authentication is available to the user. The default is "yes".

StrictHostKeyChecking

Sets how strict the client should be in order to add host keys or accept connections when the server key has changed. When the setting is "yes", the user must manually update the host key files. If "no" is selected, any host key will be added without further confirmations. The default is "ask", so the user can decide which host keys to add and which to refuse.

TCPKeepAlive

This option allows the client system to send TCP keepalive messages so it can detect when a connection has failed. The default is "yes".

User Used to set the user name to log into the SSH-2 server.

UserKnownHostsFile

Sets the path and name of the file to use for host keys. The default is c:\adx_sdt1\adx_sdt1\adxssshkh.dat.

SFTP permissions file (ADXSSHXH.DAT)

ADXSSHXH.DAT is the 4690 OS SFTP permission file for the server. The 4690 OS SFTP server (ADXSSHPL.386), uses a permission file similar to the permission file for the 4690 OS FTP server. The differences are:

- User names are system users.
- No passwords are included or needed in the permission file. Passwords are verified through a system password service.
- The permission file and path is c:\adx_sdt1\ADXSSHXH.DAT
- There is not an encrypted version of this permission file at this time.

File ADX_SDT1:ADXSSHXH.DAT must be created by the user. ADXSSHXH.DAT cannot be changed or deleted using a SSH-2 SFTP client.

The SFTP server (ADXSSHPL.386) does not require ADXSSHXH.DAT to start.

If ADXSSHXH.DAT is present, it is composed of entries that specify the user and a series of paths that the user can or cannot switch to. File permission is set at a directory basis. Each entry should have at least a user and can be followed by several path permission lines. There should be one keyword option per line. A line can be up to 80 characters long.

Sample file

```
user: 99999999
wr: c:/users c:/adx_sdt1
rd: c:/users c:/adx_sdt1 c:/adx_spgm

user: 1234567
wr^: c:/adx_spgm c:/adx_sdt1
rd^: c:/users/u1234567 c:/adx_sdt1
```

Where:

user Sets current use to apply the following path permissions until a new user entry is found. Users should be system users.

wr:, wr^:

Sets a write (wr:) or write not (wr^) permission to the path or list of paths. The paths are separated by spaces. Per user you can use wr: or wr^:, but not both. If the list of paths is longer than 80 characters, you can keep adding lines as long as you use the same permission (wr: or wr^) for the additional lines.

rd:, rd^:

Sets a read (rd:) or read not (rd^) permission to the path or list of paths. The paths are separated by spaces. Per user you can use rd: or rd^:, but not both. If the list of paths is longer than 80 characters, you can keep adding lines as long as you use the same permission (rd: or rd^) for the additional lines.

Authorizing users for SSH or SFTP

Use of SSH or SFTP or both require that the Enhanced Security function of 4690 OS be enabled. Users can be authorized for SSH, SFTP, or both. This authorization is performed by using the Authorization Manager function in the Enhanced Security system utility. Entries for "SSH Secure Remote Logon" and "SSH Secure FTP Logon" are found under the Communication Functions section.

Running ADXSSHFL (SFTP) as a background application

It is possible to execute the SFTP console session (ADXSSHFL.386) as a background application to automate file transfers and other activities. The configuration steps below demonstrate how to set up an automated SFTP session from a 4690 OS SFTP Client. For information purposes, the server steps are also given and are specific to a 4690 OS SFTP Server. The key management and user access will be different depending on which SFTP server is used.

In order to run the SFTP console session as a background task without user intervention, key authentication must be configured and used.

Setup the 4690 OS SSH/SFTP Client

1. Using the Enhanced Security System Utility, create a user with the name **BACKGRND**. This user must be given permission to use the SSH and SFTP functions (in the System Configuration section).

Note: The settings that have been configured for enhanced security passwords also apply to the BACKGRND userid. However, for the type of usage described here, an expired password will NOT cause a failure and will not interrupt automated SSH/SFTP sessions from performing their activities.

2. Create a file (for example, C:/BACKGRND.IN) containing the desired SFTP commands. See "Supported SFTP interactive commands" on page 78 for the list of valid commands and descriptions. An example of the type of commands that could be included in this file follows:

```

lcd localdir
pwd
cd adx_sdt1
get *.dat
bye

```

3. Create a batch file (for example, C:/BACKGRND.BAT) which will invoke the SFTP client as follows:
`adxsshfl -b c:/backgrnd.in userid1@10.1.1.1`

where:

adxsshfl is the call to the SFTP client application

-b c:/backgrnd.in designates the file (created in step 2) from which to read the commands, rather than reading from the console. This file can be located in any directory.

userid1@10.1.1.1 is the user@host on the SFTP server machine.

The bat file can log messages to stdout and stderr by adding **>stdout.dat >*stderr.dat** options to the SFTP client invocation.

4. Define a new background application with the following parameters:

INITIAL MESSAGE

Defined by user.

PROGRAM NAME

adx_spgm:command.286

PARAMETER LIST

-c C:\BACKGRND.BAT

IPL START

Set as appropriate.

5. Run ADXSSHL.386 to generate RSA (and/or DSA) key as shown in the following example:

```

adxsshl -t rsa -f adx_sdt1:adxsshrc.dat -C "" -N ""

```

Note: In this example, no passphrase was used for the key. Option -N "" specifies no passphrase.

This will generate 2 files, ADX_SDT1:ADXSSHRC.DAT is this client's private RSA key and ADX_SDT1:ADXSSHRC.PBK is this client's public RSA key (or ADX_SDT1:ADXSSHDC.DAT and ADX_SDT1:ADXSSHDC.PBK for DSA keys). The public key must be included in the SFTP Server's authorized client key file in order for unassisted key authentication to occur (see step 1 of "Setup the 4690 OS SSH/SFTP Server" on page 87).

6. Copy or append the server's public host key to the client's Known Hosts file, ADX_SDT1:ADXSSHKH.DAT. Then edit the file and add the IP address of the SFTP server at the beginning of the host key data. Otherwise you will get the following error message in the stdout file: The authenticity of host 10.1.1.1 cannot be established. RSA key fingerprint is XX:XX:XX:XX Are you sure you want to continue connecting (yes/no)?

Note: If the SFTP client has already run and answered (yes) to the authenticity of this host, then the SSH Client has already created an entry in the Known Hosts key file and this step can be omitted.

7. Perform the "Setup the 4690 OS SSH/SFTP Server" on page 87.
8. Run BACKGRND.BAT manually from the command line to make sure that everything is working as expected. One difference to this method is that the user currently logged in (not the BACKGRND user) must have the proper authority and permissions to perform the SFTP commands in the input file.
9. Activate Controller Configuration and IPL controller.
10. Depending on the IPL START setting in step 4 above, start the background application, if necessary.

Setup the 4690 OS SSH/SFTP Server

1. Copy/append the client's public key (ADX_SDT1:ADXSSSRC.PBK in the client example above or ADX_SDT1:ADXSSHDC.PBK) to the authorized client key file, ADX_SDT1:ADXSSHAK.DAT (or ADX_SDT1:ADXSSHBK.DAT).
2. Edit the SFTP Server Permission file, ADX_SDT1:ADXSSHXH.DAT, to add permissions for the user (userid1 in the example above) to access the necessary files/directories. The example below copied the anonymous user settings.

```
user: userid1  
rd: c:\ d:\ m:\ t:\  
wr: c:\ d:\ m:\ t:\
```

3. Activate Controller Configuration and IPL the controller.

You can also start SFTP (ADXSSHFL) from another running program using a similar procedure. In this case, however, the batch file is not needed.

Note: When the program is started from another running program, the user ID that the current program is running under is inherited by SFTP. All programs started as background tasks have BACKGRND as a user ID. Foreground programs take the ID of the user who is logged in.

Chapter 5. Using the Remote Change Management Server

This chapter describes the Remote Change Management Server (RCMS). The RCMS enables the Distributed Systems Executive (DSX) or the Tivoli® NetView® Distribution Manager (NetView DM) to interface with the operating system. The chapter assumes that you are familiar with the DSX or NetView DM and therefore describes only how the DSX or NetView DM is used with the operating system. If you need detailed information about the DSX program or the NetView DM program, refer to the appropriate DSX or NetView DM documentation.

The RCMS is a store controller program that runs as a background application. It is monitored from the Background Application Control panel. Messages are displayed that give status information while the program is executing. The RCMS performs the following functions:

- Provides the DSX or NetView DM with access to 4690 files
- Communicates over an SDLC/SNA network connecting a host processor to the store controller
- Provides error reporting and recovery for failures
- Provides data format conversion of files

The Distributed Systems Executive

The Distributed Systems Executive (DSX) is a host program used to help manage and control the distribution of software and data to 4690 Store Systems within a distributed data processing network. The DSX interfaces with the RCMS program on the operating system.

NetView Distribution Manager

NetView Distribution Manager (NetView DM) is a host program that helps manage and control the distribution of software and data to 4690 Store Systems within a distributed data processing network. NetView DM interfaces with the RCMS program on the operating system. NetView DM provides additional network management function over the DSX.

RCMS logical unit characteristics

The RCMS is addressed as an LU in a communications network and supports only LU-LU session type 0. A session with the RCMS is requested by the host sending a BIND request to the store controller for the RCMS LU. While the session is active, the DSX or NetView DM functions as the primary LU and the RCMS functions as the secondary LU. The session is ended when the host sends an UNBIND request or an unrecoverable error causes the RCMS to terminate.

An LU is reserved in the store controller for the RCMS. A default session configuration is predefined for the RCMS LU. If needed, the LU address can be changed from the default assignment.

The BIND request for the RCMS LU must specify:

- Function Management (FM) Profile 3
- Transmission Services (TS) Profile 4
- Primary and secondary LU protocols for FM data: multiple request unit (RU) chaining, immediate request mode, and definite or exception responses
- Common LU protocols for FM data: brackets not used, and duplex send/receive mode for normal-flow RUs
- Maximum RU size of 256 bytes or 512 bytes

An SDLC link must be configured at the master store controller to support the LU-LU session. The link must be active before the session with the RCMS can be started. Typically, the link is activated and monitored by the background application program, ADXHSNLL.286.

RCMS commands

This section describes the processing of the DSX or NetView DM commands by the RCMS. See *DSX Preparing and Tracking Transmission Plans* or the *NetView/Distribution Manager User's Guide* for the DSX or NetView DM commands and parameters for the operating system.

The RCMS processes DSX or NetView DM commands that enable you to:

- Add new files at the 4690 store controller
- Replace existing files
- Retrieve files or a catalog of files from the 4690 store controller
- Retrieve a list of file names and other information for a 4690 directory
- Delete files
- Send and begin execution of an operating system command list file for starting utilities or user programs
- Send and asynchronously or synchronously execute an operating system command list file for starting utilities or user programs (NetView DM only)
- Query the status of previously started asynchronous command lists (NetView DM only)
- Send a message to the operator at the 4690 store controller

Using the SEND command with a new file

The SEND command adds a new file to a directory on a 4690 store controller disk when the file does not exist. The disk can be any disk (or diskette) device on the controller connected to the host network or on any other LAN-connected controller.

The host can add only a product control file (PCF) to a program directory (for example, ADX_SPGM) to support installation of a licensed product from the host. You must use the Apply Software Maintenance (ASM) utility to send all other program directory files to the corresponding maintenance subdirectory and added to the program subdirectory. Note that you must use the SEND command and not the UPDATE command. If you use the UPDATE command, RCMS in the operating system rejects the command by sending error message 80B60721 back to NetView/DM in the host processor.

On a LAN system, you must specify a file attribute parameter that identifies the LAN characteristics of the file. You must specify a file type for data set resources, which identifies the type of data that the host file contains; either one of the following:

- binary, extended binary-coded decimal interchange code (EBCDIC)
- 4690 Programmable Store System (PSS)

See “RCMS and files” on page 94 for more information about file types.

File ownership and access rights for the new file are the same as for the subdirectory to which the file is being added.

Using the SEND command when the file already exists

The SEND command replaces files that already exist on a 4690 controller disk. The disk can be any disk (or diskette) device on the store controller connected to the host network or on any other LAN-connected controller. The file is overwritten with the new data.

The host cannot replace a file in a program directory. Instead, you must send the file to the maintenance subdirectory and replace it in the program directory using the ASM utility. See the *4690 OS: Programming Guide* for more information on the ASM utility.

The RCMS does not replace a file that is open for use by another program. For a 4690 store controller LAN system, the file distribution attributes are not changed after the file is replaced, but remain as originally set for the file.

For data set resources, you must specify a file type indicating the type of data (EBCDIC, binary, or 4690 PSS) the host file contains. See “RCMS and files” on page 94 for more information about file types.

File ownership and access rights for the new file are the same as for the existing file.

RETRIEVE command

This command retrieves files, or a list of files, from a specified directory on a store controller disk. The disk can be any disk (or diskette) device on the controller connected to the host network or any other LAN-connected store controller. The RCMS shares read access for all files requested by the RETRIEVE command, allowing a file to be sent to the host when it is open for use by another program.

For data set resources, you must specify a file type to indicate the data format (EBCDIC, binary, or 4690 PSS) that the file is to be received at the host.

Keyed files retrieved as 4690 PSS type data are received as sequential files. Only valid records are retrieved and no null records are transmitted. Records are retrieved according to physical record order within the keyed file and not according to key sequence.

All files within a 4690 directory can be retrieved by a single DSX or NetView DM command by specifying a catalog retrieval.

If the resource name specified by the RETRIEVE command is in the form *logical_dataset_name*, the RCMS examines the *file_name* and *extension* fields of the fully expanded file specification (after a name has been substituted for the *logical_dataset_name*) for global characters “*” and “?”. If any global characters are found, the RCMS sends to the host a data stream that is equivalent to the 4690 DIR command for the specified directory. The data set type parameter is ignored when the RETRIEVE command is used for DIR requests. The format of the output is:

```
Volume in drive "disk id" is "volume label"
Directory of
"node id: :disk id :\path\"

"fname" "ext" "fsize" "date changed" "time changed"
.
.
.
"number" file(s) "number" bytes free
```

For the directories defined by Toshiba, default logical data set names exist that allow retrieval of directory information.

Table 12 identifies the logical data set names and the actual file specifications for root directories.

Table 12. Directory information for root directories

Logical data set name	File specification
D_A	A:*.*
D_B	B:*.*
D_C	C:*.*
D_D	D:*.*

Table 13 on page 92 identifies the logical data set names and the actual file specifications for system directories.

Table 13. Directory information for system directories

Logical data set name	File specification
D_SPGM	ADX_SPGM:*. *
D_SMNT	ADX_SMNT:*. *
D_SBUL	ADX_SBUL:*. *
D_SDT1A	A:\ADX_SDT1*. *
D_SDT1B	B:\ADX_SDT1*. *
D_SDT1C	C:\ADX_SDT1*. *
D_SDT1D	D:\ADX_SDT1*. *

Table 14 identifies the logical data set names and the actual file specifications for application directories.

Table 14. Directory information for application directories

Logical data set name	File specification
D_IPGM	ADX_IPGM:*. *
D_IMNT	ADX_IMNT:*. *
D_IBUL	ADX_IBUL:*. *
D_IDT1A	A:\ADX_IDT1*. *
D_IDT1B	B:\ADX_IDT1*. *
D_IDT1C	C:\ADX_IDT1*. *
D_IDT1D	D:\ADX_IDT1*. *
D_IDT4A	A:\ADX_IDT4*. *
D_IDT4B	B:\ADX_IDT4*. *
D_IDT4C	C:\ADX_IDT4*. *
D_IDT4D	D:\ADX_IDT4*. *

Table 15 identifies the logical data set names and the actual file specifications for user directories.

Table 15. Directory information for user directories

Logical data set name	File specification
D_UPGM	ADX_UPGM:*. *
D_UMNT	ADX_UMNT:*. *
D_UBUL	ADX_UBUL:*. *
D_UDT1A	A:\ADX_UDT1*. *
D_UDT1B	B:\ADX_UDT1*. *
D_UDT1C	C:\ADX_UDT1*. *
D_UDT1D	D:\ADX_UDT1*. *

DELETE command

This command deletes a file from a specific subdirectory on a store controller disk. The disk can be any disk (or diskette) device on the store controller connected to the host network or any other LAN-connected store controller. The host cannot delete a file from any 4690 program directory. You must delete the file using the ASM utility.

The RCMS does not delete a file that is open for use by another program.

INITIATE FUNCTION command

This command sends and executes an operating system command list file at the store controller. The command list file contains only commands that start system utilities or user programs as background applications. Batch file (.BAT) processing cannot be done using this function. The command list file should contain a single EBCDIC record consisting of the program file specification and program parameter string. The program file specification is separated by a blank character from the parameter string. Each parameter in the parameter string is separated by a comma. The maximum program file specification length is 24 characters and the maximum parameter string length is 46 characters.

The RCMS ensures that the program file to be started exists, but does not determine completion status. If completion status is required, the program being executed must report it.

INITIATE FUNCTION command (synchronous)

The synchronous INITIATE FUNCTION command is for use with NetView DM only.

A synchronous command list allows the RCMS to keep the SNA session active until the command list has completed. Then, the RCMS returns the completion status of the command list to NetView DM whether the status is successful or unsuccessful. A successful completion is indicated by a return code of zero. An unsuccessful completion is indicated by a 4-byte negative return code from the 4690 application. NetView DM splits this 4-byte return code into a 2-byte primary and a 2-byte secondary return code.

Note: If the 4690 application ends with a positive return code, NetView DM indicates a successful completion by a return code of zero. The RCMS does not return a positive number as a return code.

You must append the string "-NDM" to the program parameter string in the command list file to signal that the RCMS must wait for the return code before acknowledging the INITIATE FUNCTION command.

For example:

```
ADX_SPGM:PROGRAM1.286 PARM1,PARM2 -NDM
```

The "-NDM" are not included in the limit of 46 characters for parameter strings because RCMS deletes them before the program is started. If you do not append the "-NDM" string, RCMS responds to this command as the "INITIATE FUNCTION command."

INITIATE FUNCTION command (asynchronous)

The asynchronous INITIATE FUNCTION command is for use with NetView DM only.

An asynchronous command list does not necessarily cause the RCMS to keep the SNA session active. The RCMS only acknowledges to NetView DM that an asynchronous command has been received but does not wait for the command to complete. Later, the RCMS returns the completion status of the command (either successful or unsuccessful) to NetView DM. See the "QUERY command (NetView DM only)" on page 94 for more information.

You start an asynchronous command using the NetView DM INITIATE FUNCTION command with SYNC=NO specified.

INFORM OPERATOR command

This command sends a message from the host to the store controller operator. The message is displayed on the 4690 system message panel.

The message should be limited to 106 EBCDIC characters. Messages longer than 106 characters are truncated.

QUERY command (NetView DM only)

The QUERY command is used to obtain the completion status of command lists that have been started asynchronously. Unlike other RCMS commands, QUERY is not explicitly initiated by the user. NetView DM issues the QUERY command automatically when a session ends and at predefined time intervals.

When all synchronous NetView DM commands scheduled on a node for the current SNA sessions are terminated, NetView DM requests the completion status of any outstanding asynchronous command lists by the QUERY command. If an asynchronous command has completed, the RCMS sends a QUERY response to NetView DM including the completion status (either successful or unsuccessful) along with the appropriate correlators such as phase name, phase date, and time. The process is repeated until a negative response to the QUERY command is received indicating that no asynchronous commands have completed. After receiving this response, NetView DM ends the SNA session.

NetView DM establishes an SNA session with the RCMS at predefined time intervals and issues a QUERY command to request the results of command lists that were started asynchronously. Refer to the INITIATE FUNCTION command in the NetView DM manuals for details. The response to the QUERY command includes the information described in the previous paragraph.

A successful completion is indicated by a return code of zero. An unsuccessful completion is indicated by a four-byte negative return code from the 4690 application. NetView DM splits this four-byte return code into a two-byte primary and a two-byte secondary return code.

Note: If the 4690 application ends with a positive return code, NetView DM indicates a successful completion by a return code of zero. The RCMS does not return a positive number as a return code. A return code of X'87654321' indicates the asynchronous command list was interrupted by an IPL of the 4690 store controller.

RCMS and files

On a LAN system, you must specify file distribution attributes for new files unless files are being replaced. The attributes that you can specify are:

- Local
- Mirrored—Distribute At Close
- Mirrored—Distribute On Update
- Compound—Distribute At Close
- Compound—Distribute On Update

These file attributes are described in the *4690 OS: Programming Guide*.

You specify the LAN attribute parameter using the DSX PREPARE utility for files originating at the host processor that are distributed to store controllers. For files originating at a 4690 store controller for distribution to other store controllers, the RCMS passes the file attribute parameter when the file is retrieved by the host, and the DSX or NetView DM maintains the file for distributed copies.

Unlike the Host Command Processor (HCP), the RCMS does not allow partial file transmission and thus, does not support intermittent transmission of sequential files. The HCP and the RCMS can exist on the same store system and can run concurrently, provided adequate memory exists. The two programs, however, cannot have concurrent access to the same file for replace or delete functions.

The RCMS does not permit file processing by record.

File recovery

The RCMS provides for recovery from interrupted file transmission without retransmitting complete files. The RCMS can recover files transmitted as binary or EBCDIC. The RCMS cannot recover 4690 PSS data. These files must be completely retransmitted. Non-user errors are logged in the 4690 system event log.

Data format file conversion

The RCMS provides for data format conversion of files for use at both the 4690 store controller and the host. The data format is from the host perspective and can be one of the following:

- Binary
- EBCDIC
- 4690 PSS

For 4690 files originating at the host to be distributed to store controllers, you specify the data format type parameter using the DSX PREPARE utility. For 4690 files at the store controller to be retrieved to the host, you specify the parameter on the RETRIEVE DATASET command.

Binary data format

This data format is compatible for the 4690 store controller without any data conversion. Binary data files are created on the store controller and distributed to other 4690 store controllers. Examples include executable programs and files that contain PC record and field separators. The fields are PC string or numeric constants.

EBCDIC data format

This data format consists of variable length records containing all EBCDIC characters whose record delimiters are carriage return/line feed (CR/LF) characters. The RCMS converts an EBCDIC data set sent by the host to ASCII data for viewing at the 4690 store controller. Conversely, ASCII data from the store controller can be requested as EBCDIC data for viewing at the host site. Examples of this data format include files such as e-mail and 4690-formatted dumps. The ASCII-EBCDIC character conversion is according to the system translation table, ADXCSOTF.DAT.

4690 Programmable Store System data format

The Programmable Store System (PSS) data format is defined by an application whose responsibility it is to convert the data during the transmission. The RCMS interfaces to this application during file transmission, sending it unconverted data records and receiving in return converted data records.

RCMS translation interface

The RCMS translation interface is similar to that of the HCP. The translation interface for the HCP is described in “The HCP Translation Interface” on page 123. The exceptions are that the RCMS does not support IBM 3650 data formats, the inbound pipe to the RCMS is named PI:ADXHSRIN, and the outbound pipe is named PI:ADXHSROT. The RCMS passes these pipe names to the translation program in the parameter string when the application is started.

Host access to store controller files

4690 files that are transmitted using the DSX or NetView DM are named according to specific conventions. The file name specification must be one of the following forms:

- *logical_device.file_name.extension*
- *logical_device.file_name*
- *logical_dataset_name*

where:

logical_device

is the logical name for the disk device name and path name to the actual file on the master store controller. This name can be from 1 to 8 alphanumeric characters, including the special characters # \$ @ & _ . (The catalog name for the RETRIEVE command is specified only as a *logical device*.)

file_name

is the actual name of the file in a file directory. The name can be from 1 to 8 alphanumeric characters, including the special characters # \$ @ & _.

extension

is the actual file name extension of the file. The name can be from 1 to 3 alphanumeric characters, including the special characters # \$ @ & _.

logical_dataset_name

is the logical name for the complete file specification including the LAN node name, device name, path name, file name, and extension. This name can be from 1 to 8 alphanumeric characters, including the special characters # \$ @ & _ . For files transmitted using the DSX, this naming convention must be used for files on controllers other than the master store controller.

Chapter 6. Using the Host Command Processor

This chapter introduces the Host Command Processor (HCP) and explains its usage.

Introduction to the HCP

The HCP is the part of the operating system that supports the management of files from a host site. It communicates with Advanced Data Communications for Stores (ADCS) as a Toshiba store controller type. It also communicates with any host application program that is written to adhere to the command protocols defined in this chapter. The HCP allows the host processor to perform such functions as:

- Sending host-prepared data files to the store controller
- Retrieving programs and data files from the store controller for distribution to other store locations
- Retrieving data files from the store controller for report processing and generation at the host
- Retrieving a list of the names and attributes of all files of a subdirectory
- Maintaining keyed files by reading, updating, adding, and deleting specific records
- Erasing files that are not needed at the store controller

ADCS provides the host with the additional capability to start system utilities such as the Keyed File utility and user-written programs.

Intermittent Transmission of PC-Format Sequential Files

Several sequential files in the store system are used to accumulate data throughout the day. To retrieve this data for the host processor, you can wait until the store has stopped operations for the day, and then retrieve all the data from each file. However, if you want to retrieve any file data as it is being accumulated during the day while the store is open, and the file is a PC format, use the following procedures:

1. Retrieve all data that has been written in the file thus far. In the first dump request, set the relative starting sector to value X'0000' and request the entire file. When a dump echo is returned to the host, bytes 58 and 59 of the echo contain the value for the relative starting sector of the next dump request.
2. Save the received file data. You should save the file data received from the store to prevent it from being replaced when a request for any new data is made. As new data is received, you should merge it with this saved data to build the entire file.
3. Retrieve all new data that has been written in the file since the last dump request. In subsequent dump requests, set the relative starting sector to the value returned in the last dump echo. Again, request the entire file, which retrieves any new data written to the file since the last dump. Bytes 58 and 59 of the dump echo contain the next starting sector value.
4. Merge the new data with the file data already received. The first data block received from a dump request for any new data can contain up to 256 bytes of data that has already been received as well as new data. You should merge this data in a way that does not result in duplicate data records. For example, suppose that the data already received and the data received from the dump request for new data is in files of 256-byte blocks. See Figure 4 on page 98 for an illustration of how you would merge the two files.
5. Determine when no more data will be accumulated for the file. After you retrieve all of the data, purge the file at your convenience.

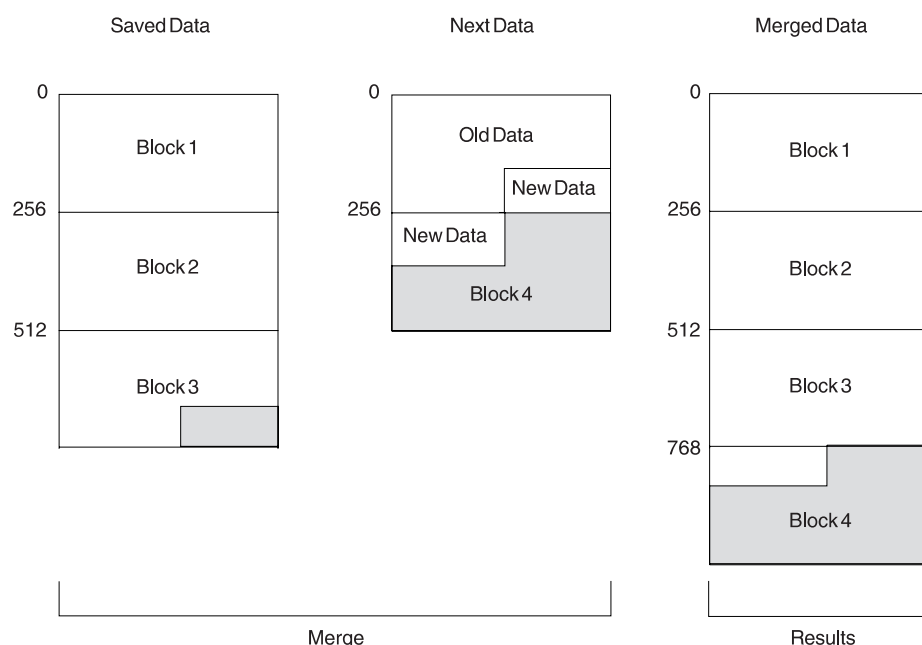


Figure 4. File Data Received from the HCP

Intermittent Transmission of PSS-Format Sequential Files

Use the following procedure to retrieve data as it is being accumulated in a PSS format sequential file. The procedure is valid only for a file of the name *pppTRANS.DAT*. An example of this type of file is a transaction summary log.

1. Retrieve all data that has been logged so far. In the first dump request, set the relative starting sector to value X'0000' and request the entire file. When the dump echo is returned to the host, bytes 58 and 59 contain a *token* value, to be used as the relative starting sector of the next dump request.
2. Save the received file data. You should save the file data received from the store to prevent its being replaced when a request for any new data is made. As new data is received, you should merge it with this saved data to build the entire file.
3. Retrieve all new data that has been logged since the last dump request. In subsequent dump requests, set the relative starting sector to the token value returned in the last dump echo. Again, request the entire file, which retrieves only the new data logged since the last dump. Bytes 58 and 59 of the dump echo contain the next token value.
4. Append the new data to the data already received. The new data received contains only the new records that were logged since the last dump request. Determine the end of the data last received and append the new data to it.
5. Determine when no more data will be logged. After you retrieve all of the data, purge the file at your convenience.

Logical Unit Characteristics

The HCP logical unit (LU) is controlled entirely from the host processor and requires no operator intervention at the store controller. The operator can monitor the operation of the HCP from the Background Application Control panel. The single available host session is activated from the host processor by a BIND command once a link has been activated to the store controller. A series of

commands is then sent to the store controller. Finally, the session is deactivated from the host processor with an UNBIND command. The HCP displays messages that indicate whether it is waiting for the host or is processing a command.

Optionally, the host or the operator can initiate the HCP from the Background Application Control panel. The HCP opens an SNA link to the host and waits until a host program requests a session. Whether the HCP is initiated by the host or by the operator as a background application, only a single session is allowed. Commands and data are exchanged between the host and the HCP in the form of SNA request units (RUs). Command RUs are distinguished from data RUs in that command RUs are formatted RUs having the SNA format indicator set in the request header for the RU. All RUs used to communicate with the HCP can be up to 256 bytes.

A command to the HCP consists of one command RU and optionally one or more data RUs. Commands can be sent as chains where the command RU is the first (or only) RU in the chain and the last data RU is the end RU in the chain. A session can consist of many command chains. If commands are not sent as chains, then the last RU of the command must have the change direction indicator set in its request/response header (RH), or it must require a definite response to the RU. The use of chaining for commands is specified in the BIND request which activates the session with the HCP.

HCP Commands

The commands in this section allow you to maintain 4690 files from the host processor. The ADCS START USER PROGRAM command (SUP), which is used in starting 4690 background applications, is described in “START USER PROGRAM (SUP) Command” on page 211.

On a LAN system, when the host retrieves files from a store controller, the file distribution attributes are not retrieved with the file. You can retrieve these file attributes by using the DUMP FILE command for the subdirectory containing the file. See the “DUMP FILE Command” on page 102 for a description of data retrieved for the subdirectory. For additional information on the operating system files, see the *4690 OS: Programming Guide*.

In the following commands, the file ID must be six characters and must be defined according to requirements in “HCP File Naming Conventions” on page 109. The password defines file ownership by specifying the user ID and group ID for the file. Specify a value of zero (0) for both IDs to default file ownership to the subdirectory. See the *4690 OS: Programming Guide* for a description of user ID and group ID.

Note: The file size limit is 8 MB unless you are using the extended dump function.

ADD KEYED RECORDS Command

The host program can add keyed records to a file by supplying the HCP with the file name and the byte count of one record (key and data), followed by RUs containing the key and data. The key must be in the same relative position and of the same length in each RU as originally defined for the file. There should be one data unit for each record to be added. All records with this command must be the same size. If varied sizes exist, an ADD KEYED RECORDS command must be sent for each unique size, followed by the key and data.

Using the add/replace option causes the HCP to add the record if it does not exist in the file and to replace the record if it does exist in the file, without returning any error status to the host.

Table 16 illustrates the format for adding keyed records.

Table 16. ADD KEYED RECORDS Command Format

Bytes	Field	Value
0	Command	X'06'

Table 16. ADD KEYED RECORDS Command Format (continued)

Bytes	Field	Value
1	Add option	X'80'
	Add/Replace option	X'82'
4-5	Type	X'0281'
22-27	File ID	C'----'
56-57	Byte count of key	X'----'

CREATE FILE Command

The CREATE FILE command requests that a new file be created by the HCP. The file type of the new file can be either sequential or keyed. The host program must specify the file name and other parameters to the HCP.

Files that are randomized as keyed at the host and require data translation are created as keyed. However, the store controller Keyed File utility must be run against the file after it is received at the store controller to make it a keyed file. Keyed files that have been previously created in the store controller and are being distributed from the host to other store controllers must be created and loaded as sequential files to allow byte stream loading. The host program must specify the file name and other parameters to the HCP.

To transmit files on a LAN system, you must specify a file type and mode for distribution. Use the VOL=*n* parameter to specify the file type. You can specify three types of files:

- 0** Local (non-distributed). The default is 0.
- 2** Mirrored.
- 3** Compound.

Use the SHARE parameter to specify whether you want the file distributed per update or at close. You have two choices with the SHARE parameter:

SHARE=no Distribute files per update. The default is no.

SHARE=yes Distribute files at close.

Table 17 describes the format of the CREATE FILE command.

Table 17. CREATE FILE Command Format

Bytes	Field	Value	Additional Information
0	Command	X'06'	
1	Option	X'80'	
4-5	Type	X'0181'	
14-15	Password	X'----'	
22-27	File ID	C'----'	
56-57	File length	(Length in 256-byte blocks)	
60	Options	Byte 60:	
		Bit 0	Multiple-use authorized (for LAN system only)
		Bit 1	Physical sequential
		Bit 2	Reserved
		Bit 3	Keyed access

Table 17. CREATE FILE Command Format (continued)

Bytes	Field	Value	Additional Information
		Bit 4	Reserved
		Bit 5	Reserved
		Bit 6	Reserved
		Bit 7	Reserved
61	Options	Byte 61:	
		Bit 0	Reserved
		Bit 1	Reserved
		Bit 2	Reserved
		Bit 3	Reserved
		Bit 4	Reserved
		Bit 5	Reserved
		Bit 6	Reserved
		Bit 7	Variable-length record format (records have 1-byte count)
65	Volume	X'00'	Any volume (for LAN system only)
100-108	Variable fields depending on file type		

Sequential Access

Byte	Field	Value
100-101	Record length if fixed	Must be less than or equal to 256 bytes.
	Maximum record length not including byte count if variable	Must be greater than 0 and less than or equal to 255 bytes.

Keyed Access

Byte	Field	Value
100-101	Record length if fixed	Must be greater than 0 and less than or equal to 254 bytes.
	Maximum record length if variable length	Must be greater than 0 and less than or equal to 253 bytes.
102-103	Key length	Must be greater than 0 and less than or equal to 254 bytes if fixed, or must be greater than 0 and less than or equal to 253 bytes if variable.
104-105	Key offset	Offset within record. Must be less than record length.
106-107	Randomizing divisor	Should be an odd number less than data set length, except if a multiple of 5 less 2.
108	Chaining threshold	Should be greater than 0. Suggested value is 4.

Notes:

1. If no password is entered, the HCP assigns default file ownership of the subdirectory the file is being created in.
2. If the default file ownership is not correct for the intended use of the file at the store controller, you can change it by specifying a password on a subsequent LOAD FILE command or the owning program must change it through the file interface request.

DELETE KEYED RECORDS Command

To request that the HCP delete keyed records, the host program must supply the file name in the command RU followed by data RUs containing only the key. The key must be in the same relative position within the RU and of the same length as originally defined for the file. One RU must exist for each record to be deleted following the command RU.

Using the Delete Unconditional option causes the HCP to avoid returning an error status to the host if the record does not exist.

Table 18 describes the format of the DELETE KEYED RECORDS command.

Table 18. DELETE KEYED RECORDS Command Format

Bytes	Field	Value
0	Command	X'06'
1	Delete option	X'80'
	Delete unconditional option	X'82'
4-5	Type	X'0283'
22-27	File ID	C'----'

DUMP FILE Command

The DUMP FILE command requests the HCP to send a file to the host. The HCP supports intermittent transmission of sequential files in either PC-format or PSS-format.

See “Intermittent Transmission of PC-Format Sequential Files” on page 97 and “Intermittent Transmission of PSS-Format Sequential Files” on page 98 for more information on the intermittent transmission of sequential files.

Partial file dumping support allows a sector (256-byte data block) relative to the beginning of the file to be specified as the first sector to be dumped. Partial file dumping support also lets you set the number of sectors to be dumped. The relative starting sector can be the first sector or any sector within the bounds of the file. The number of sectors to be dumped can be any number that does not request more data than that contained in the file (from the specified starting sector to the end of the file). If the file size is not known at the host, then specifying the number of sectors to be dumped as a negative number, as zero (0), or as X'7FFF' causes the HCP to dump data until it reaches the end of the file.

A dump echo RU is always returned in response to the DUMP FILE command prior to the file data. The dump echo RU is an echo of the dump command received by the HCP with the option field set to X'83'. If the number of sectors requested was a negative number, zero, or X'7FFF', the HCP also sets the number of sectors requested field, EOD sector offset, and EOD byte offset fields according to the actual amount of data dumped.

The dump echo for PSS mode files reflects the physical size of the file in sectors as it exists on the 4690 system if the number of sectors specified on the dump request is zero. The size returned is not an indication of the amount of data to be returned because the HCP cannot estimate the actual amount of data to be dumped on a PSS mode file. Therefore, the receiving application must be able to accept the data sent without regard to the value returned in the dump echo.

A list of all files contained in a subdirectory can be retrieved from the host using the DUMP FILE command by specifying a reserved name for the file name that is recognized by the HCP as a subdirectory name. See “HCP File Naming Conventions” on page 109 for the reserved names for subdirectories. The data returned to the host by the HCP is a list of the files in the subdirectory. Each file in the subdirectory is described by a 48-byte entry with the format shown in Figure 5 on page 103.

	0	1	2	3
0	N/A			
4	NAME			
8				
12				
16				
20	ATTRIB			
24	RECSIZE		USER	GROUP
28	PROTECT		RESERVED	
32	RESERVED		RESERVED	
36	SIZE			
40	MODYEAR		MODMONTH	MODDAY
44	MODHR	MODMIN	MODSEC	RESERVED

48 = Length in bytes, where:

NAME = Disk file name (EBCDIC).

The variable length name is terminated by a byte of binary zeros.

ATTRIB = The sum of the following file attributes&colon.

Figure 5. Subdirectory File List Description

Table 19 shows the value of each byte in a non-LAN system.

Table 19. Non-LAN System Bytes

Byte	Value
First Byte	
X'00'	Reserved
X'01'	Reserved
X'02'	Reserved
X'04'	Reserved
X'08'	Reserved
X'20'	Reserved
X'40'	Reserved
X'60'	Reserved
X'80'	Reserved
X'A0'	Reserved
X'C0'	Reserved
X'E0'	Reserved
Second Byte	
X'01'	Read-only

Table 19. Non-LAN System Bytes (continued)

Byte	Value
X'02'	Hidden
X'04'	System
X'08'	Volume label
X'10'	Subdirectory
X'20'	Archive
X'40'	Reserved
X'80'	Reserved

Table 20 shows the value of each byte in a LAN system.

Table 20. LAN System Bytes

Byte	Value
First Byte	
X'00'	Local/Update
X'01'	Reserved
X'02'	Reserved
X'04'	Reserved
X'08'	Reserved
X'20'	Local/Close
X'40'	Mirrored/Update
X'60'	Compound/Update
X'80'	Not Used
X'A0'	Not Used
X'C0'	Mirrored/Closed
X'E0'	Compound/Close
Second Byte	
X'01'	Read-only
X'02'	Hidden
X'04'	System
X'08'	Volume label
X'10'	Subdirectory
X'20'	Archive
X'40'	Security enabled *
X'80'	Reserved

*: Used only for volume label.

Table 21 describes the file attributes of the data returned to the host by the HCP.

Table 21. Returned Data File Attributes

Attribute	Description
RECSIZE	The record size expressed as a numeric value. *

Table 21. Returned Data File Attributes (continued)

Attribute	Description
USER	The owner's user ID.
GROUP	The owner's group ID.
PROTECT	The file security access rights. **
SIZE	The file size expressed as a numeric value. *
MODYEAR	The year of the last modification. This is a numeric value. *
MODMONTH	The month of the last modification (1-12).
MODDAY	The day of the last modification (1-31).
MODHR	The hour of the last modification (0-23).
MODMIN	The minute of the last modification (0-59).
MODSEC	The second of the last modification (0-59).

*: All numeric values are expressed from the least significant byte (LSB) to the most significant byte (MSB) in left-to-right order. For example, X'0001' is equal to 256 decimal.

**: See the 4690 OS: Programming Guide for a description of file security.

In the dump echo returned for a dump directory command, the following items have no meaning, and are set to binary zero:

- EOD byte offset
- EOD sector offset
- Number of sectors requested
- Relative starting sector

Table 22 describes the format of the DUMP FILE command.

Table 22. DUMP FILE Command Format

Bytes	Field	Value	Additional Information
0	Command	X'03'	
1	Option	X'03'	X'83' in this field indicates a dump echo RU from the HCP.
4-5	Type	X'0400'	
22-27	File ID	C'----'	File name or reserved name for subdirectory.
54-55	Starting point in file	X'----'	Relative sector offset for PC format files or starting token for PSS format files.
56-57	Number of sectors requested	X'----'	If necessary, the HCP resets this field in the dump echo RU to the actual number of sectors to be transmitted.
58-59	EOD sector offset	X'----'	Returned in the dump echo RU for sequential files.
60-61	EOD Byte Offset	X'----'	Returned in the dump echo RU for sequential files.

The HCP EXTENDED DUMP Command

The HCP EXTENDED DUMP command RU supports 4690 files greater than 32K sectors (a sector is 256 bytes). The HCP uses the EXTENDED DUMP Command/Echo fields instead of, but for the same purposes as, the corresponding fields of the normal DUMP Command/Echo when processing the DUMP command. The EXTENDED DUMP command is not available to ADCS users.

Table 23 describes the format of the EXTENDED DUMP command.

Table 23. EXTENDED DUMP Command Format

Bytes	Field	Value	Additional Information
0	Command	X'03'	
1	Option	X'03'	X'83' in this field indicates a dump echo request unit from the HCP.
2-3	N/A		
4-5 *	Type	X'0401'	Extended dump option.
6-21	N/A		
22-27	File ID	C'----'	File name or reserved name for subdirectory.
28-39	N/A		
40-43 *	Starting point in file	X'----'	Extended format: relative sector offset to start dump.
44-47 *	Number of sectors requested	X'----'	Extended format: number of sectors to be dumped. If value X'7FFFFFFF' or X'00000000', then the HCP is to dump to EOF and return actual number of sectors to be dumped.
48-51 *	EOD sector offset	X'----'	Extended format: EOD sector offset returned by the HCP.
52-59 *	N/A		
60-61	EOD byte offset	X'----'	EOD byte offset returned by the HCP.

Note: * Field that differs from the normal DUMP command.

LOAD FILE Command

The host program uses the LOAD FILE command to load or replace a specified file on the store controller disk. It can either load the entire file or replace part of a PC-format file. If the file is in use at the store, the HCP rejects the load request. The file is truncated to the sector value specified by the relative starting sector prior to loading the data.

The length of the file after the LOAD FILE command has been completed is based on the amount of data received. For PC-format files, it is the starting sector value plus the length of the data received. For PSS-format files, it is the resulting length of the translated data.

Table 24 describes the format of the LOAD FILE command.

Table 24. LOAD FILE Command Format

Bytes	Field	Value	Additional Information
0	Command	X'01'	
1	Initial	X'80'	
	Replace option	X'82'	
2-3	N/A		
4-5	Type	X'0400'	
		X'0600'	Data truncation. Not valid for PC-format files.
6-13	N/A		
14-15	Password	X'----'	See Note 2.
16-21	N/A		

Table 24. LOAD FILE Command Format (continued)

Bytes	Field	Value	Additional Information
22-27	File ID	C'-----'	
28-49	N/A		
54-55	Relative starting sector	X'----'	See Note 1.
56-57	Number of sectors	X'----'	See Note 1.

Note:

1. Valid only for files loaded in PC format.
2. The password parameter is used to change the ownership of the file.

PURGE FILE Command

The host program can request the HCP to delete a file from the store controller disk with the PURGE FILE command. If the file is in use at the store, then the HCP rejects the purge request.

Table 25 describes the format of the PURGE FILE command.

Table 25. PURGE FILE Command Format

Bytes	Field	Value
0	Command	X'06'
1	Option	X'80'
4-5	Type	X'0183'
22-27	File ID	C'-----'

READ KEYED RECORD Command

The host program can request the HCP to retrieve a specified keyed record from a file and transmit it to the host. The host application program must supply the file name in the command RU, followed by a second RU containing the key.

The key must be in the same relative position within the RU and of the same length as originally defined for the file.

The store controller returns an echo of the command RU containing the length of the keyed record in bytes 56 and 57. This echo is followed by a second RU with the key and the record that was read. The store controller also sends status response information, after it receives a response, to the RU that contained the record.

Table 26 describes the format of the READ KEYED RECORD command.

Table 26. READ KEYED RECORD Command Format

Bytes	Field	Value
0	Command	X'06'
1	Option	X'01'
4-5	Type	X'0284'
22-27	File ID	C'-----'

REPLACE KEYED RECORDS Command

The host program can also request the HCP to replace keyed records. It must send the file name to the HCP in the command RU, followed by data RUs containing the key and the data. The key must be in the same relative position within each RU and of the same length as originally defined for the file. One RU must exist for each record to be replaced.

If records of different lengths are sent, a separate command RU must be used for each record length that is sent. Using the add/replace option causes the HCP to add the record if it is not already in the file. If the record is already in the file, the HCP replaces it without returning error status to the host.

Table 27 describes the format of the REPLACE KEYED RECORDS command.

Table 27. REPLACE KEYED RECORDS Command Format

Bytes	Field	Value
0	Command	X'06'
1	Replace option or Add/Replace option	X'80' X'82'
4-5	Type	X'0282'
22-27	File ID	C'----'

STATUS Command

The host program uses the STATUS command to request the status of the previous command. The host program can send a STATUS command after any command series in which it sends data to the HCP. The HCP sends a status response request unit in reply to a STATUS command and also after any request for file data from the store controller (for example, DUMP FILE and READ KEYED RECORD commands).

Table 28 describes the format of the STATUS command.

Table 28. STATUS Command Format

Bytes	Field	Value
0	Command	X'05'
1	Option	X'81'

Status Response Format from the HCP to the Host

The HCP sends status information on the last command completed to the host program in a status RU at the following times:

- In reply to a STATUS command from the host. (Length of 30 bytes. However, only the first 22 bytes are valid.)
- At the completion of data transfer to the host as a result of a DUMP FILE command. (Length of 30 bytes. However, only the first 8 bytes are valid.)
- At the completion of any ACTION command in which the HCP sends data to the host. For example, after a READ KEYED RECORD command. (Length of 30 bytes. However, only the first 8 bytes are valid.)

Table 29 on page 109 describes the format for the status response information that the HCP sends to the host application program.

Table 29. Status Response Format

Bytes	Field	Value
0	Command	X'05'
1	Valid Option codes: Complete Error	X'81' X'C0'
2	Error code	First byte of user sense field:
2	(if applicable)	X'01' – Data validity error X'02' – Resources unavailable X'07' – I/O error X'08' – File full X'09' – File not found X'0C' – Duplicate file or member name X'10' – Invalid starting address X'11' – Invalid ending disk address X'3F' – Undefined
7	User error code (if applicable)	Last byte of user sense field
8	Command code (From LOAD FILE command for which status information is given)	X'03' – Dump X'05' – Status X'06' – Action
9	Option code (From command for which status information is given)	X'80' – Initial option X'81' – Add option X'82' – Replace option X'83' – Delete option
10 - 11	Type code	X'0181' – CREATE FILE X'0281' – ADD KEYED RECORDS X'0282' – REPLACE KEYED RECORDS X'0283' – DELETE KEYED RECORDS X'0284' – READ KEYED RECORD X'0400' – LOAD or DUMP FILE X'401' – EXTENDED DUMP (see Note 1 on page 113)
12 - 17	File ID	C'-----'
18 - 19	Blanks	
20 - 21	Last sector saved	X'----'

If the status response is to an EXTENDED DUMP command, the first 24 bytes are valid and the last sector saved field is in positions 20 - 23. The EXTENDED DUMP function is not available to ADCS users.

HCP File Naming Conventions

The HCP/ADCS interface supports the use of a 6-character name to identify the files to be accessed in the operating system. For LAN systems, all files accessed through the HCP are on the master store controller unless otherwise noted.

Note: File names greater than 8.3 characters in length are not supported through HCP. You must use the Remote Command Processor (RCP) to access files greater than 8.3 characters in length. See Chapter 7, “Using the Remote Command Processor,” on page 131 for more information.

To allow the 6-character name to reference a fully qualified file name in the operating system, special characters are used in the 6-character name, and some restrictions are placed on the file names that can be used. The file name restrictions are described in the *4690 OS: Programming Guide*.

The HCP converts the 6-character file name as shown in Table 30 on page 110.

Table 30. Subdirectory (Dump of List of Files in the Named Subdirectory)

6-Character Name	Operating System File Name Built by the HCP from 6-Character Name
RDIR1:	Root directory on C drive
RDIR2:	Root directory on A drive
RDIR3:	Root directory on third drive (B, D, or O)
SDIRL*	ADX_SPGM subdirectory
SDIR!	ADX_SMNT subdirectory
SDIR1/	ADX_SDT1 subdirectory on C drive
SDIR2/	ADX_SDT1 subdirectory on A drive
SDIR3/	ADX_SDT1 subdirectory on third drive (B, D, or O)
*SDIRL	ADX_IPGM subdirectory
!SDIRL	ADX_IMNT subdirectory
/SDIR1	ADX_IDT1 subdirectory on C drive
/SDIR2	ADX_IDT1 subdirectory on A drive
/SDIR3	ADX_IDT1 subdirectory on third drive (B, D, or O)
(SDIR1	ADX_IDT4 subdirectory on C drive
(SDIR2	ADX_IDT4 subdirectory on A drive
(SDIR3	ADX_IDT4 subdirectory on third drive (B, D, or O)
SDIR*L	ADX_UPGM subdirectory
SDIR!L	ADX_UMNT subdirectory
SDIR/1	ADX_UDT1 subdirectory on C drive
SDIR/2	ADX_UDT1 subdirectory on A drive
SDIR/3	ADX_UDT1 subdirectory on third drive (B, D, or O)
SDIRx:	User-defined subdirectory (see Note 1 on page 113)

System Files

Table 31 contains information about system files.

Note: Files in ADX_SPGM can only be read (dumped) from the host.

Table 31. Naming System Files

6-Character Name	Operating System File Name Built by the HCP from the 6-Character Name	Type of File	Actual File Name (filename.ext)
abcdL*	ADX_SPGM:ADXabcdL.286	PC	ADXabcdL.286
abcdE*	ADX_SPGM:ADXabcdE.EXE	PC	ADXabcdE.EXE
abcdV*	ADX_SPGM:ADXabcdV.OVR	PC	ADXabcdV.OVR
abcdY*	ADX_SPGM:ADXabcdY.SYS	PC	ADXabcdY.SYS
abcdF*	ADX_SPGM:ADXabcdF.DAT	PC	ADXabcdF.DAT
abcdI*	ADX_SPGM:ADXabcdI.DAT	PC	ADXabcdI.DAT
abcdS*	ADX_SPGM:ADXabcdS.DAT	PC	ADXabcdS.DAT
abcdZ*	ADX_SPGM:ADXabcdZ.BAT	PC	ADXabcdZ.BAT
abcdD*	ADX_SPGM:ADXabcdD.DAT	PC	ADXabcdD.DAT
abcdW*	ADX_SPGM:ADXabcdW.SRL	PC	ADXabcdW.SRL

Table 31. Naming System Files (continued)

6-Character Name	Operating System File Name Built by the HCP from the 6-Character Name	Type of File	Actual File Name (filename.ext)
abcdU*	ADX_SPGM:ADXabcdL.BSX	PC	ADXabcdL.BSX
CbTdU*	ADX_SPGM:ADXCbTdU.DAT	PC	ADXCbTdU.DAT
CbTdU!	ADX_SMNT:ADXCbTdU.DAT	PC	ADXCbTdU.DAT
abcdL!	ADX_SMNT:ADXabcdL.286	PC	ADXabcdL.286
abcdE!	ADX_SMNT:ADXabcdE.EXE	PC	ADXabcdE.EXE
abcdV!	ADX_SMNT:ADXabcdV.OVR	PC	ADXabcdV.OVR
abcdY!	ADX_SMNT:ADXabcdY.SYS	PC	ADXabcdY.SYS
abcdF!	ADX_SMNT:ADXabcdF.DAT	PC	ADXabcdF.DAT
abcdI!	ADX_SMNT:ADXabcdI.DAT	PC	ADXabcdI.DAT
abcdS!	ADX_SMNT:ADXabcdS.DAT	PC	ADXabcdS.DAT
abcdZ!	ADX_SMNT:ADXabcdZ.BAT	PC	ADXabcdZ.BAT
abcdK!	ADX_SMNT:ADXabcdK.L86	PC	ADXabcdK.L86
abcdD!	ADX_SMNT:ADXabcdD.DAT	PC	ADXabcdD.DAT
abcdW!	ADX_SMNT:ADXabcdW.SRL	PC	ADXabcdW.SRL
abcdU!	ADX_SMNT:ADXabcdL.BSX	PC	ADXabcdL.BSX
ADXxx:	ADXxx	PC	(see Note 2 on page 113)
abcdF/	ADX_SDT1:ADXabcdF.DAT	PC	ADXabcdF.DAT
abcdF&	ADX_SDT1:ADXabcdF.DAT	PRINT	ADXabcdF.DAT
abcdF%	ADXabcdF (see Note 6 on page 113)	PC	ADXabcdF.DAT
abcdF\$	ADXabcdF (see Note 6 on page 113)	PRINT	ADXabcdF.DAT

Point-of-Sale Application Files

Table 32 contains information about point-of-sale application files. In the table, the lowercase letters correspond to characters in a logical file name. The variable *ppp* corresponds to the application prefix that is defined through configuration.

Note: Files in ADX_IPGM can only be read (dump) from the host.

Table 32. Naming Point-of-Sale Application Files

6-Character Name	Operating System File Name Built by the HCP from the 6-Character Name	Type of File	Actual File Name (filename.ext)
*bcdeL	ADX_IPGM:pppbcdL.286	PC	pppbcdL.286
*bcdeV	ADX_IPGM:pppbcdV.OVR	PC	pppbcdV.OVR
*bcdeF	ADX_IPGM:pppbcdF.DAT	PC	pppbcdF.DAT
*bcdel	ADX_IPGM:pppbcdL.DAT	PC	pppbcdL.DAT
*bcdeS	ADX_IPGM:pppbcdS.DAT	PC	pppbcdS.DAT
*bcdeU	ADX_IPGM:pppbcdL.BSX	PC	pppbcdL.BSX
*bcdeZ	ADX_IPGM:pppbcdZ.BAT	PC	pppbcdZ.BAT
*bcdeD	ADX_IPGM:pppbcdD.DAT	PC	pppbcdD.DAT
!bcdel	ADX_IMNT:pppbcdL.286	PC	pppbcdL.286
!bcdeS	ADX_IMNT:pppbcdS.DAT	PC	pppbcdS.DAT

Table 32. Naming Point-of-Sale Application Files (continued)

6-Character Name	Operating System File Name Built by the HCP from the 6-Character Name	Type of File	Actual File Name (filename.ext)
!bcdeV	ADX_IMNT:pppbcdV.OVR	PC	pppbcdV.OVR
!bcdeF	ADX_IMNT:pppbcdF.DAT	PC	pppbcdF.DAT
!bcdeU	ADX_IMNT:pppbcdL.BSX	PC	pppbcdL.BSX
!bcdeK	ADX_IMNT:pppbcdK.L86	PC	pppbcdK.L86
!bcdel	ADX_IMNT:pppbcdL.DAT	PC	pppbcdL.DAT
!bcdeZ	ADX_IMNT:pppbcdZ.BAT	PC	pppbcdZ.BAT
!bcdeD	ADX_IMNT:pppbcdD.DAT	PC	pppbcdD.DAT
#bcdeL	ADX_IPGM:ADXbcdL.286	PC	ADXbcdL.286
#bcdeV	ADX_IPGM:ADXbcdV.OVR	PC	ADXbcdV.OVR
#bcdeF	ADX_IPGM:ADXbcdF.DAT	PC	ADXbcdF.DAT
#bcdel	ADX_IPGM:ADXbcdL.DAT	PC	ADXbcdL.DAT
#bcdeS	ADX_IPGM:ADXbcdS.DAT	PC	ADXbcdS.DAT
#bcdeU	ADX_IPGM:ADXbcdL.BSX	PC	ADXbcdL.BSX
#CcTeU	ADX_IPGM:ADXCcTeU.DAT	PC	ADXCcTeU.DAT
#bcdeZ	ADX_IPGM:ADXbcdZ.BAT	PC	ADXbcdZ.BAT
#bcdeD	ADX_IPGM:ADXbcdD.DAT	PC	ADXbcdD.DAT
?bcdeL	ADX_IMNT:ADXbcdL.286	PC	ADXbcdL.286
?bcdeV	ADX_IMNT:ADXbcdV.OVR	PC	ADXbcdV.OVR
?bcdeF	ADX_IMNT:ADXbcdF.DAT	PC	ADXbcdF.DAT
?bcdel	ADX_IMNT:ADXbcdL.DAT	PC	ADXbcdL.DAT
?bcdeS	ADX_IMNT:ADXbcdS.DAT	PC	ADXbcdS.DAT
?bcdeU	ADX_IMNT:ADXbcdL.BSX	PC	ADXbcdL.BSX
?CcTeU	ADX_IMNT:ADXCcTeU.DAT	PC	ADXCcTeU.DAT
?bcdeZ	ADX_IMNT:ADXbcdZ.BAT	PC	ADXbcdZ.BAT
?bcdeD	ADX_IMNT:ADXbcdD.DAT	PC	ADXbcdD.DAT
/bcdef	ADX_IDT1:pppbcdF.DAT	PC	pppbcdF.DAT
&bcdef	ADX_IDT1:pppbcdF.DAT	PRINT	pppbcdF.DAT
(bcdef	ADX_IDT4:pppbcdF.DAT	PC	pppbcdF.DAT
)bcdef	ADX_IDT4:pppbcdF.DAT	PRINT	pppbcdF.DAT
%bcdef	pppbcdF (see Note 6 on page 113)	PC	(see Note 3 on page 113)
\$bcdef	pppbcdF (see Note 6 on page 113)	PRINT	(see Note 3 on page 113)
+bcdef	pppbcdF (see Note 6 on page 113)	new PSS	(see Note 3 on page 113)
abcdef	pppbcdF (see Note 6 on page 113)	old PSS	(see Note 3 on page 113)
=bcdef	UUUbcdef (see Note 6 on page 113)	PC	(see Note 4 on page 113)
-bcdef	UUUbcdef (see Note 6 on page 113)	PRINT	(see Note 4 on page 113)
>bcdef	UUUbcdef (see Note 6 on page 113)	new PSS	(see Note 4 on page 113)
<bcdef	UUUbcdef (see Note 6 on page 113)	old PSS	(see Note 4 on page 113)

User Files

Table 33 contains information about user files. In the table, the lowercase letters correspond to characters in a logical file name. The variable *xxx* is determined by the File Use table. See Table 34 on page 114 for information about the File Use table. The variable *zzz* is determined by the Translation table. See Table 35 on page 114 for information about the Translation table.

Note: Files in ADX_UPGM can only be read (dump) from the host.

Table 33. Naming User Files

6-Character Name	Operating System File Name Built by the HCP from the 6-character Name	Type of File	Actual File Name (filename.ext)
abcd*w	ADX_UPGM: <i>abcd.xxx</i>	PC	<i>abcd.xxx</i>
abcd!w	ADX_UMNT: <i>abcd.xxx</i>	PC	<i>abcd.xxx</i>
abcd/y	ADX_UDT1: <i>abcd.zzz</i>	See Table 35 on page 114	<i>abcd.zzz</i>
abcd(y	ADX_UDT2: <i>abcd.zzz</i>	See Table 35 on page 114	<i>abcd.zzz</i>
abcd)y	ADX_UDT3: <i>abcd.zzz</i>	See Table 35 on page 114	<i>abcd.zzz</i>
abcde:	<i>abcde</i> (see Note 6)	PC	(see Note 5)

Notes for Using the Tables to Name HCP Files

1. The subdirectory used for the list function must be defined by a define statement for SDIRx where x is an alphanumeric character. These define statements are specified through configuration. For example, SDIRA could be defined to be C:\DELIVERY\. For LAN systems, definition of SDIR should include the store controller ID.
2. The real file name for the ADXxx files are defined by a define statement for ADXxx. The ADXxx names are used for Operating System Command Mode files that have required file names such as BACKUP, CHKDSK, COMP, DISKCOMP, and DISKCOPY. For a list of these names, see "Special Logical Names" on page 118.
3. The actual file name for the pppb_cdef files are defined by the application logical names file that is provided with the application diskettes. The logical name of the file is *pppb_cdef*.
For the names that can be used, refer to the section of your sales application guide that describes host data transfer.
4. The actual file name for the UUUb_cdef files is defined by the application logical names file. The fully qualified file name can be defined through configuration panels. The logical name of the file must be *uuub_cdef*. For more information on defining fully qualified file names, see the *4690 OS: Programming Guide*.
5. The actual file name for abcde files is defined by the user logical names file. The fully qualified file name can be defined through configuration panels. The logical name of the file must be *abcde*. For more information on fully qualified file names, see the *4690 OS: Programming Guide*.
6. For LAN systems, the definition of the logical file name built by the HCP defines the store controller the file exists on.

File Use Table

Table 34 is used for determining variables for user files. The File Use table affects the first two rows in Table 33 on page 113. A value from the **w** column in the translation table replaces the *w* variable in the 6-character name in Table 33 on page 113, and a value from the **xxx** column replaces the *xxx* variable in the operating system file name column.

Table 34. File Use Table

w	xxx	Intended File Usage
A	A86	Assembler source code
B	BAS	4680 BASIC source code Note: In the ADX_SPGM and ADX_SMNT subdirectories, B is used for font files and uses FNT to replace the <i>xxx</i> variable.
C	C	C source code
D	DAT	Product Control File
E	EXE	Executable files
F	DAT	Messages, Input Sequence Tables, and similar files
G		Reserved, Do not use
H	H	C source code Include files
I	DAT	Product Control File Extension
J	J86	4680 BASIC source code Include Files
K	L86	Library files
L	286	4690-executable file
M	MAP	Linkage Editor Map file
N	INP	Linkage Editor Input file
O	OBJ	Relocatable Object file
P	LST	Language Translator Listing file
Q	LIS	4690 Interlisting file
R	LIN	4690 Line Number file
S	DAT	DisplayManager* Screen files
T	SYM	Linkage Editor Symbol Table file
U	BSX	Symbols file
V	OVR	4690-executable Overlay file
W	SRL	Shareable Runtime libraries
X	XRF	Cross-Reference file
Y	SYS	4690 non-relocatable file
Z	BAT	Batch files

Translation Table

Table 35 is used for determining variables for user files. The translation table affects the last three rows in Table 33 on page 113. A value from the **y** column in the translation table replaces the *y* variable in the 6-character name in Table 33 on page 113, and a value from the **zzz** column replaces the *zzz* variable in the operating system file name column.

Table 35. Translation Table

y	zzz	File Type
A	ASC	PRINT

Table 35. Translation Table (continued)

y	zzz	File Type
B	BIN	PC
X	XLT	new PSS or old PSS

PC format files do not need to be translated.

The translation for new PSS and old PSS is by a user-provided program. See “The HCP Translation Interface” on page 123 and “RCMS translation interface” on page 95 for a description of user program capabilities.

The translation for print files is from ASCII to EBCDIC or from EBCDIC to ASCII. The ASCII file records have a format that represents a print line. The ASCII record length must be less than 255 characters and must be terminated with CR/LF characters. The EBCDIC record begins with a one-byte length field and the CR/LF characters are not present. If the byte length field is 0, it is interpreted as a CRLF.

The translation is performed as shown in Table 36.

Table 36. ASCII-EBCDIC Translation Table for Print Files

Character	ASCII Value	EBCDIC Value
(Tab Key)	009	X'05'
(Line Feed Key)	010	X'0A'
(Home Key)	011	X'0B'
(Form Feed Key)	012	X'0C'
(Carriage Return Key)	013	X'0D'
(Space Bar)	032	X'40'
!	033	X'5A'
"	034	X'7F'
#	035	X'7B'
\$	036	X'5B'
%	037	X'6C'
&	038	X'50'
'	039	X'7D'
(040	X'4D'
)	041	X'5D'
*	042	X'5C'
+	043	X'4E'
,	044	X'6B'
-	045	X'60'
.	046	X'4B'
/	047	X'61'
0	048	X'F0'
1	049	X'F1'
2	050	X'F2'
3	051	X'F3'

Table 36. ASCII-EBCDIC Translation Table for Print Files (continued)

Character	ASCII Value	EBCDIC Value
4	052	X'F4'
5	053	X'F5'
6	054	X'F6'
7	055	X'F7'
8	056	X'F8'
9	057	X'F9'
	058	X'7A'
;	059	X'5E'
<	060	X'4C'
=	061	X'7E'
>	062	X'6E'
?	063	X'6F'
@	064	X'7C'
A	065	X'C1'
B	066	X'C2'
C	067	X'C3'
D	068	X'C4'
E	069	X'C5'
F	070	X'C6'
G	071	X'C7'
H	072	X'C8'
I	073	X'C9'
J	074	X'D1'
K	075	X'D2'
L	076	X'D3'
M	077	X'D4'
N	078	X'D5'
O	079	X'D6'
P	080	X'D7'
Q	081	X'D8'
R	082	X'D9'
S	083	X'E2'
T	084	X'E3'
U	085	X'E4'
V	086	X'E5'
W	087	X'E6'
X	088	X'E7'
Y	089	X'E8'
Z	090	X'E9'
[091	X'AD'

Table 36. ASCII-EBCDIC Translation Table for Print Files (continued)

Character	ASCII Value	EBCDIC Value
\	092	X'E0'
]	093	X'BD'
^	094	(undefined)
_	095	X'6D'
	096	X'79'
a	097	X'81'
b	098	X'82'
c	099	X'83'
d	100	X'84'
e	101	X'85'
f	102	X'86'
g	103	X'87'
h	104	X'88'
i	105	X'89'
j	106	X'91'
k	107	X'92'
l	108	X'93'
m	109	X'94'
n	110	X'95'
o	111	X'96'
p	112	X'97'
q	113	X'98'
r	114	X'99'
s	115	X'A2'
t	116	X'A3'
u	117	X'A4'
v	118	X'A5'
w	119	X'A6'
x	120	X'A7'
y	121	X'A8'
z	122	X'A9'
{	123	X'C0'
	124	X'6A'
}	125	X'D0'
~	126	X'A1'

Note: Characters not listed in this table are considered “undefined” for print format files. Undefined characters are translated to the underscore character (_).

HCP File Security

Files created with the HCP are assigned user ID, group ID, and access rights. User ID and group ID are obtained from the password values on the HCP CREATE FILE and LOAD FILE commands. Access privileges are controlled by the HCP according to Table 36 on page 115. A “1” means that the access privilege is allowed and a “0” means that the access privilege is not allowed. See the *4690 OS: Programming Guide* for more information about File Security.

Table 37. HCP File Security

File in Subdirectory/File	Default User ID	Default Group ID	World RWED	Group RWED	Owner RWED
ADX_SPGM	1	1	1 0 1 0	1 0 1 0	1 1 1 1
ADX_SMNT	1	1	1 0 1 0	1 0 1 0	1 1 1 1
ADX_SDT1	1	1	1 0 1 0	1 0 1 0	1 1 1 1
ADX_IPGM	1	2	0 0 0 0	1 0 1 0	1 1 1 1
ADX_IMNT	1	2	0 0 0 0	1 0 1 0	1 1 1 1
ADX_IDT1	1	2	0 0 0 0	1 0 1 0	1 1 1 1
ADX_IDT4	1	2	0 0 0 0	1 0 1 0	1 1 1 1
ADX_UPGM	1	3	0 0 0 0	1 0 1 0	1 1 1 1
ADX_UMNT	1	3	0 0 0 0	1 0 1 0	1 1 1 1
ADX_UDT1	1	3	0 0 0 0	1 0 1 0	1 1 1 1
Others	1	3	0 0 0 0	1 0 1 0	1 1 1 1

Note: Access privileges are referred to in the table as follows: R=Read, W=Write, E=Execute, D=Delete

Special Logical Names

Table 38 contains logical names that do not follow normal ADX naming conventions.

Table 38. Special Logical Names

Logical Name	Related File	File Definition
ADX00:	ADX_SPGM:COMMAND.286	Command Shell
ADX01:	ADX_SMNT:COMMAND.286	Command Shell
ADX02:	ADX_SPGM:COPY.286	File Copy Utility
ADX03:	ADX_SMNT:COPY.286	File Copy Utility
ADX04:	ADX_SPGM:FDISK.286	Disk Partition Utility
ADX05:	ADX_SMNT:FDISK.286	Disk Partition Utility
ADX06:	ADX_SPGM:FORMAT.286	Disk Format Utility
ADX07:	ADX_SMNT:FORMAT.286	Disk Format Utility
ADX08:	ADX_SPGM:MKDIR.286	Subdirectory Utility
ADX09:	ADX_SMNT:MKDIR.286	Subdirectory Utility
ADX0A:	ADX_SPGM:RENAME.286	File Rename Utility
ADX0B:	ADX_SMNT:RENAME.286	File Rename Utility
ADX0C:	ADX_SPGM:TYPE.286	Display File Utility
ADX0D:	ADX_SMNT:TYPE.286	Display File Utility
ADX0E:	ADX_SPGM:MORE.286	Display File One Screen at a Time Utility
ADX0F:	ADX_SMNT:MORE.286	Display File One Screen at a Time Utility

Table 38. Special Logical Names (continued)

Logical Name	Related File	File Definition
ADX10:	ADX_SPGM:ATHD.286	Fixed Disk Driver
ADX11:	ADX_SMNT:ATHD.286	Fixed Disk Driver
ADX12:	ADX_SPGM:PRINTER.286	Printer Driver
ADX13:	ADX_SMNT:PRINTER.286	Printer Driver
ADX14:	ADX_SPGM:BACKUP.286	Disk Backup Utility
ADX15:	ADX_SMNT:BACKUP.286	Disk Backup Utility
ADX16:	ADX_SPGM:CHKDSK.286	Disk Statistics Utility
ADX17:	ADX_SMNT:CHKDSK.286	Disk Statistics Utility
ADX18:	ADX_SPGM:COMP.286	File Compare Utility
ADX19:	ADX_SMNT:COMP.286	File Compare Utility
ADX1A:	ADX_SPGM:DISKCOMP.286	Disk Compare Utility
ADX1B:	ADX_SMNT:DISKCOMP.286	Disk Compare Utility
ADX1C:	ADX_SPGM:DIR.286	List Directory Files Utility
ADX1D:	ADX_SMNT:DIR.286	List Directory Files Utility
ADX1E:	ADX_SPGM:DISKCOPY.286	Disk Copy Utility
ADX1F:	ADX_SMNT:DISKCOPY.286	Disk Copy Utility
ADX20:	ADX_SPGM:DISKSET.286	Disk Protect Utility
ADX21:	ADX_SMNT:DISKSET.286	Disk Protect Utility
ADX22:	ADX_SPGM:DREDIX.286	Editor
ADX23:	ADX_SMNT:DREDIX.286	Editor
ADX24:	ADX_SPGM:FIND.286	Find String Utility
ADX25:	ADX_SMNT:FIND.286	Find String Utility
ADX26:	ADX_SPGM:FSET.286	File Attributes Utility
ADX27:	ADX_SMNT:FSET.286	File Attributes Utility
ADX28:	ADX_SPGM:PRINT.286	Display Print Spooler Status Utility
ADX29:	ADX_SMNT:PRINT.286	Display Print Spooler Status Utility
ADX2A:	ADX_SPGM:RESTORE.286	Disk Restore Utility
ADX2B:	ADX_SMNT:RESTORE.286	Disk Restore Utility
ADX2C:	ADX_SPGM:RMDIR.286	Subdirectory Utility
ADX2D:	ADX_SMNT:RMDIR.286	Subdirectory Utility
ADX2E:	ADX_SPGM:SID.286	Debug Program
ADX2F:	ADX_SMNT:SID.286	Debug Program
ADX30:	ADX_SPGM:SORT.286	Line Sort Utility
ADX31:	ADX_SMNT:SORT.286	Line Sort Utility
ADX32:	ADX_SPGM:TREE.286	Directory Path Display
ADX33:	ADX_SMNT:TREE.286	Directory Path Display
ADX34:	ADX_SPGM:VER.286	Version Display
ADX35:	ADX_SMNT:VER.286	Version Display
ADX36:	ADX_SPGM:VOL.286	Disk Volume Label Display
ADX37:	ADX_SMNT:VOL.286	Disk Volume Label Display

Table 38. Special Logical Names (continued)

Logical Name	Related File	File Definition
ADX38:	ADX_SPGM:TEXT	Sample Edit File
ADX39:	ADX_SMNT:TEXT	Sample Edit File
ADX3A:	ADX_SPGM:HELP.EDX	Editor Help File
ADX3B:	ADX_SMNT:HELP.EDX	Editor Help File
ADX3C:	ADX_SPGM:LSN.EDX	Editor Tutorial File
ADX3D:	ADX_SMNT:LSN.EDX	Editor Tutorial File
ADX3E:	ADX_SPGM:STUDENT.BAT	Editor Tutorial
ADX3F:	ADX_SMNT:STUDENT.BAT	Editor Tutorial
ADX42:	ADX_SPGM:CONFIG.286	Serial Port Utility
ADX43:	ADX_SMNT:CONFIG.286	Serial Port Utility
ADX44:	ADX_SPGM:DATE.286	Set Date Utility
ADX45:	ADX_SMNT:DATE.286	Set Date Utility
ADX46:	ADX_SPGM:DESPOOL.286	Despooler Utility
ADX47:	ADX_SMNT:DESPOOL.286	Despooler Utility
ADX48:	ADX_SPGM:DMED.286	Display Manager Editor
ADX49:	ADX_SMNT:DMED.286	Display Manager Editor
ADX4A:	ADX_SPGM:DMEDCOL.286	Display Manager Editor
ADX4B:	ADX_SMNT:DMEDCOL.286	Display Manager Editor
ADX4C:	ADX_SPGM:DMSAMPLE.286	Display Manager Sample
ADX4D:	ADX_SMNT:DMSAMPLE.286	Display Manager Sample
ADX50:	ADX_SPGM:LOGON.286	OS LOGON Command
ADX51:	ADX_SMNT:LOGON.286	OS LOGON Command
ADX52:	ADX_SPGM:PASSWORD.286	Change User Password
ADX53:	ADX_SMNT:PASSWORD.286	Change User Password
ADX54:	ADX_SPGM:PROCESS.286	Process Utility
ADX55:	ADX_SMNT:PROCESS.286	Process Utility
ADX56:	ADX_SPGM:RECDIR.286	Subdirectory Utility
ADX57:	ADX_SMNT:RECDIR.286	Subdirectory Utility
ADX58:	ADX_SPGM:RECFILE.286	Disk File Utility
ADX59:	ADX_SMNT:RECFILE.286	Disk File Utility
ADX5A:	ADX_SPGM:SPOOL.286	Print Spool Utility
ADX5B:	ADX_SMNT:SPOOL.286	Print Spool Utility
ADX5C:	ADX_SPGM:SYS.286	System Transfer Utility
ADX5D:	ADX_SMNT:SYS.286	System Transfer Utility
ADX5E:	ADX_SPGM:TIME.286	Set Time Utility
ADX5F:	ADX_SMNT:TIME.286	Set Time Utility
ADX60:	ADX_SPGM:WMEX.286	Window Manager
ADX61:	ADX_SMNT:WMEX.286	Window Manager
ADX62:	ADX_SPGM:NETDEV.286	Network Device Driver
ADX63:	ADX_SMNT:NETDEV.286	Network Device Driver

Table 38. Special Logical Names (continued)

Logical Name	Related File	File Definition
ADX64:	ADX_SPGM:NSD.286	Network Name Service Driver
ADX65:	ADX_SMNT:NSD.286	Network Name Service Driver
ADX66:	ADX_SPGM:XPORT.286	Network Transport Driver
ADX67:	ADX_SMNT:XPORT.286	Network Transport Driver
ADX68:	ADX_SPGM:CDOSLDR.286	Second Stage Loader
ADX69:	ADX_SMNT:CDOSLDR.286	Second Stage Loader
ADX6E:	ADX_SPGM:NETDEV.BSX	Network Transport Driver
ADX6F:	ADX_SMNT:NETDEV.BSX	Network Transport Driver
ADX70:	ADX_SPGM:NSD.BSX	Network Transport Driver
ADX71:	ADX_SMNT:NSD.BSX	Network Transport Driver
ADX76:	ADX_SPGM:ADXHSZ1L.EXE	ASYNCR Driver
ADX77:	ADX_SMNT:ADXHSZ1L.EXE	ASYNCR Driver
ADX7A:	ADX_SPGM:FD.286	Diskette Driver
ADX7B:	ADX_SMNT:FD.286	Diskette Driver
ADX7C:	ADX_SPGM:FD.BSX	Diskette Driver
ADX7D:	ADX_SMNT:FD.BSX	Diskette Driver
ADX7E:	ADX_SPGM:ADXHSM0E.EXE	RIC SDLC Support
ADX7F:	ADX_SMNT:ADXHSM0E.EXE	RIC SDLC Support
ADX80:	ADX_SPGM:ADXHS50E.EXE	RIC X.25 Support
ADX81:	ADX_SMNT:ADXHS50E.EXE	RIC X.25 Support
ADX82	ADX_SPGM:RECOVER.BAT	Restore Original NLS Translate Tables
ADX83	ADX_SMNT:RECOVER.BAT	Restore Original NLS Translate Tables
ADX84	ADX_SPGM:TAPESTRM.BAT	Tape Streamer Device Driver Selection
ADX85	ADX_SMNT:TAPESTRM.BAT	Tape Streamer Device Driver Selection
ADX86	ADX_SPGM:ADXHSM0E.MAP	SDLC Code MAP File
ADX87	ADX_SMNT:ADXHSM0E.MAP	SDLC Code MAP File
ADX88	ADX_SPGM:ADXHS50E.MAP	X.25 Code MAP File
ADX89	ADX_SMNT:ADXHS50E.MAP	X.25 Code MAP File
ADX8A	ADX_SPGM:DISP_SCR.286	Display a Screen From a BAT File
ADX8B	ADX_SMNT:DISP_SCR.286	Display a Screen From a BAT File
ADX8C	ADX_SPGM:SHELLSRL.SRL	Shared Runtime Library
ADX8D	ADX_SMNT:SHELLSRL.SRL	Shared Runtime Library
ADX8E	ADX_SPGM:UTILMSG.SRL	Shared Runtime Library
ADX8F	ADX_SMNT:UTILMSG.SRL	Shared Runtime Library
ADX90	ADX_SPGM:UTOOL_SB.SRL	Shared Runtime Library
ADX91	ADX_SMNT:UTOOL_SB.SRL	Shared Runtime Library
ADX92	ADX_SPGM:ADXACRBW.BSX	Controller BASIC Shared Runtimes
ADX93	ADX_SMNT:ADXACRBW.BSX	Controller BASIC Shared Runtimes
ADX94	ADX_SPGM:ADXACRBW.SYM	Controller BASIC Shared Runtimes
ADX95	ADX_SMNT:ADXACRBW.SYM	Controller BASIC Shared Runtimes

Table 38. Special Logical Names (continued)

Logical Name	Related File	File Definition
ADX96	ADX_SPGM:ADXHSDMG.DAT	Host Configuration File
ADX97	ADX_SMNT:ADXHSDMG.DAT	Host Configuration File
ADX98	ADX_SPGM:ADXCP437.DAT	Display Character Sets
ADX99	ADX_SMNT:ADXCP437.DAT	Display Character Sets
ADX9A	ADX_SPGM:ADXCP850.DAT	Display Character Sets
ADX9B	ADX_SMNT:ADXCP850.DAT	Display Character Sets
ADX9C	ADX_SPGM:ADXCP852.DAT	Display Character Sets
ADX9D	ADX_SMNT:ADXCP852.DAT	Display Character Sets
ADX9E	ADX_SPGM:ADXCP857.DAT	Display Character Sets
ADX9F	ADX_SMNT:ADXCP857.DAT	Display Character Sets
ADXA0	ADX_SPGM:ADXCP863.DAT	Display Character Sets
ADXA1	ADX_SMNT:ADXCP863.DAT	Display Character Sets
ADXA2	ADX_SPGM:ADXCP865.DAT	Display Character Sets
ADXA3	ADX_SMNT:ADXCP865.DAT	Display Character Sets
ADXA4	ADX_SPGM:ADXCP869.DAT	Display Character Sets
ADXA5	ADX_SMNT:ADXCP869.DAT	Display Character Sets
ADXA6	ADX_SPGM:ADXNS000.DAT	Country-specific Host Translate Tables
ADXA7	ADX_SMNT:ADXNS000.DAT	Country-specific Host Translate Tables
ADXA8	ADX_SPGM:ADXNS037.DAT	Country-specific Host Translate Tables
ADXA9	ADX_SMNT:ADXNS037.DAT	Country-specific Host Translate Tables
ADXAA	ADX_SPGM:ADXNS200.DAT	Country-specific Host Translate Tables
ADXAB	ADX_SMNT:ADXNS200.DAT	Country-specific Host Translate Tables
ADXAC	ADX_SPGM:ADXNS275.DAT	Country-specific Host Translate Tables
ADXAD	ADX_SMNT:ADXNS275.DAT	Country-specific Host Translate Tables
ADXAE	ADX_SPGM:ADXNS300.DAT	Country-specific Host Translate Tables
ADXAF	ADX_SMNT:ADXNS300.DAT	Country-specific Host Translate Tables
ADXB0	ADX_SPGM:ADXNS337.DAT	Country-specific Host Translate Tables
ADXB1	ADX_SMNT:ADXNS337.DAT	Country-specific Host Translate Tables
ADXB2	ADX_SPGM:ADXNS500.DAT	Country-specific Host Translate Tables
ADXB3	ADX_SMNT:ADXNS500.DAT	Country-specific Host Translate Tables
ADXB4	ADX_SPGM:ADXNS577.DAT	Country-specific Host Translate Tables
ADXB5	ADX_SMNT:ADXNS577.DAT	Country-specific Host Translate Tables
ADXB6	ADX_SPGM:ADXNS700.DAT	Country-specific Host Translate Tables
ADXB7	ADX_SMNT:ADXNS700.DAT	Country-specific Host Translate Tables
ADXB8	ADX_SPGM:ADXNS737.DAT	Country-specific Host Translate Tables
ADXB9	ADX_SMNT:ADXNS737.DAT	Country-specific Host Translate Tables
ADXBBA	ADX_SPGM:ADXNS773.DAT	Country-specific Host Translate Tables
ADXBBA	ADX_SMNT:ADXNS773.DAT	Country-specific Host Translate Tables
ADXBBC	ADX_SPGM:ADXNS777.DAT	Country-specific Host Translate Tables
ADXBBD	ADX_SMNT:ADXNS777.DAT	Country-specific Host Translate Tables

Table 38. Special Logical Names (continued)

Logical Name	Related File	File Definition
ADXB	ADX_SPGM:ADXNS778.DAT	Country-specific Host Translate Tables
ADXB	ADX_SMNT:ADXNS778.DAT	Country-specific Host Translate Tables
ADXC0	ADX_SPGM:ADXNS784.DAT	Country-specific Host Translate Tables
ADXC1	ADX_SMNT:ADXNS784.DAT	Country-specific Host Translate Tables
ADXC2	ADX_SPGM:ADXNS785.DAT	Country-specific Host Translate Tables
ADXC3	ADX_SMNT:ADXNS785.DAT	Country-specific Host Translate Tables
ADXC4	ADX_SPGM:ADXNS797.DAT	Country-specific Host Translate Tables
ADXC5	ADX_SMNT:ADXNS797.DAT	Country-specific Host Translate Tables
ADXC6	ADX_SPGM:ADXNS800.DAT	Country-specific Host Translate Tables
ADXC7	ADX_SMNT:ADXNS800.DAT	Country-specific Host Translate Tables
ADXC8	ADX_SPGM:ADXNS826.DAT	Country-specific Host Translate Tables
ADXC9	ADX_SMNT:ADXNS826.DAT	Country-specific Host Translate Tables
ADXC	ADX_SPGM:ADXNS900.DAT	Country-specific Host Translate Tables
ADXC	ADX_SMNT:ADXNS900.DAT	Country-specific Host Translate Tables
ADXC	ADX_SPGM:ADXNS975.DAT	Country-specific Host Translate Tables
ADXC	ADX_SMNT:ADXNS975.DAT	Country-specific Host Translate Tables
ADXD0	ADX_SPGM:ADXHSZ1L.MAP	ASYN MAP File
ADXD1	ADX_SMNT:ADXHSZ1L.MAP	ASYN MAP File
ADXD2	ADX_SPGM:REST4680.286	4680 Version of the Restore Command
ADXD3	ADX_SMNT:REST4680.286	4680 Version of the Restore Command
ADXD8	ADX_IDT1:ADXMICRF.DAT	MICR Format File

Table 39 contains other 6-character names that do not follow normal ADX naming conventions.

An x represents a system name and *abcd* represents the system file name characters that are used in the HCP name.

Table 39. 6-Character Names That Do Not Follow ADX Naming Conventions

HCP Name	System File Name	Description
CSLTF/	ADX_SDT1:ADXCSTLF.DAT	Terminal Dump
RT1SU*	ADX_SPGM:ADXRT1SL.BSX	Terminal BSX
CSLCF%	C:\ADXCSTLCF.DAT	Controller Dump
abcdB%	ADX_SPGM:xxxabcdx.BSX	Controller BSX
abcdM%	ADX_SMNT:xxxabcdx.BSX	Controller BSX

The HCP Translation Interface

The operating system uses HCP translation interfaces to communicate between the translation application and the HCP.

Pipe Interface

Two pipes exist for the communications between the translation application and the HCP. One of the pipes is for inbound messages to the HCP and the other is for outbound messages. Both of the pipes are

created by the HCP when it becomes aware that the host system is requesting access to a file on the controller's disk that need translating to new or old PSS format.

The HCP then loads the translation application and starts it. The name that is used by the HCP for the translation application is ____HSHTP, where "____" is the 3-character licensed product prefix defined during configuration. It is a logical name that is defined to a .286 file.

The translation application must open the inbound pipe to the HCP for write only and the outbound pipe from the HCP for read only. If the translation application has an error that causes termination, the pipes are automatically closed by the operating system.

The inbound pipe to the HCP is named PI:ADXHSHIN. The outbound pipe from the HCP is named PI:ADXHSHOT.

Message Format

Every message that comes to or from the translation application follows a format that consists of a record header, a data area, and always ends with a two-byte CR/LF sequence. The record header is 9 bytes long and the meaning of each of the bytes is described in Table 40. The data area contains the actual data to be sent to the HCP or received from the HCP. The translation application should read the data area from the pipe using the length field of the record header. Reading by the CR/LF delimiter is not recommended because the data area might have the binary representation of a CR/LF before the end of the record. The pipe system would then satisfy the delimited read without the complete record.

Record Header

The record header consists of nine bytes that describe the data which follows the header in the pipe. Table 40 describes the specific use of each of the bytes.

Table 40. Record Header for HCP Translation Interface

Byte	Description
0	End of Record (EOR) indicator. This byte specifies if the data area which is in the pipe is the end of a logical record. If records are too large to fit in one data area, they can be split into several data areas. The first portions have the EOR byte set OFF (0), and only the last portion of the record has the EOR byte set ON (1).
1	End of Group (EOG) indicator. The HCP always sets this byte. This byte specifies if the record which is in the pipe concludes a group of file records. If the HCP sends one record to the translation application and the translation application would like to send back more than one record, the EOG byte should be set OFF for all but the last record in that group. The EOR byte must always be ON if the EOG byte is ON because the EOG indicator is only set once for each group. If the translation application receives a record from the HCP and does not have a record to send back, it can send an EOG with an empty data area and the HCP sends another record. This can be used to skip useless records or to combine more than one record.
2	End of File (EOF) indicator. This byte specifies the end of all of the file data to be processed for the command. When EOF is sent, the data area is empty but a CR/LF sequence still follows the record header. When the translation application receives an EOF from the HCP, it can send back more records if needed. After any records are sent, the EOF header with an empty data area is then sent back to the HCP.
3	Direction indicator. This byte specifies if the record being processed is one in which the information flows from the host to the store controller or from the store controller to the host. For example, if a DUMP command is issued, the records are to be sent from the store controller to the host. The direction of the flow in this case is from store controller to host.
4	Record Level Command (RLC) indicator. This byte tells the translation application if the command from the host is at a record level (ADD, REPLACE, DELETE, or READ keyed record), in which case only individual records are being processed, or file level, in which case an entire file is being processed from start to finish.

Table 40. Record Header for HCP Translation Interface (continued)

Byte	Description
5	Key Only indicator. This byte specifies if the data area contains a key for a keyed record or if it contains a record. This indicator is set only if the RLC byte is set.
6	Indicator Byte. This is a multi-function byte which can be used to send any kind of indication between the HCP and the translation application. It is currently used as a method for the translation application to report an error to the HCP. It is also used by the HCP to accompany the file name to indicate if the file is stored at the host in the old PSS format or in the new PSS data format.
7	Message Length low-order byte. The message length is the length of the data area in bytes plus 2 bytes for the CR/LF sequence. In the case of an empty data area as with an EOF indicator, the message length should still be 2 for the CR/LF sequence.
8	Message Length high-order byte. See description of byte 7.

Data Area

The data area portion of the message in the pipe contains the actual data to be processed. This area has two different maximum sizes based on the direction that the information is to flow. If a message is going to the host (for example, a DUMP FILE command), the data area can be 256 bytes long. If the message is going to the store controller (for example, a LOAD FILE command), the data area can be up to 512 bytes long.

Format Types

From the format information that the translation application sends to the HCP, the HCP can split apart and combine the blocks of data from or to the host by records. The HCP ensures that the proper byte counts or CR/LFs are added based on the record types.

Figure 6 on page 126 illustrates the formats of the records at the store controller and the host.

Host formats (256 Byte Sectors)

OLD PSS FORMATS

- cc - 1 - data - 1 - data - ... - 00000...00000 - variable / keyed
- cc - data - data - data - ... - 00000...00000 - fixed / keyed
- 1 - data - 1 - data - 1 - data - ... - 00...0 - variable / sequential
- data - data - data - data - ... - 0000...00000 - fixed / sequential

NEW DATA FORMAT (→ cr/lf)

- data - e - data - e - data - e - data... - variable / sequential

4690 formats (512 Byte Sectors)

- data - e - data - e - data - e - data... - variable / sequential
- data - data - data - data - data - data... - fixed / sequential
- cc - data - data - data - data - data - data - 0...0 - keyed / only fixed

Figure 6. Record Formats at the Store Controller and at the Host

End of File

After this exchange of file name and format information, records or keys are passed to the translation application for translation and are then passed back to the HCP. When the HCP has no more records to be sent, a final message is sent with the EOF byte in the record header set on. The translation application then either responds with more data or with the EOF itself.

If no records have been translated, no messages can be returned to the HCP after the EOF notification except EOF. If any records were translated and there is more data, all of that data should be sent before the EOF indication is returned.

Note that on a record-level command, no data records can be returned after the EOF indication from the HCP. Only an EOF can be returned to the HCP.

After all data has been sent, the EOF message is sent with the EOF byte in the record header set ON. The format of this last message both from the HCP and from the translation application:

1	1	1	0	0	0	0	0	2	CR	LF
---	---	---	---	---	---	---	---	---	----	----

Figure 7 on page 127 shows examples of the kinds of information that can flow between the HCP and the translation application.

Translation Interface

TRANSLATION INTERFACE DATA STRUCTURE

Record header	Data area	CR LF
---------------	-----------	----------

\ 9 bytes / \ 256 bytes or 512 bytes /

HCP Request		Application Response	
Direction, format, (key info)	PC filename	EOG	Format info
		Error	Error info
Direction and (EOR)	file record	(EOR)/(EOG)	xlate file record
		EOG	(need more)
		Error	Error info
Direction and 'Key Only'	record key	EOG	xlate key
		Error	Error info
EOF		(EOR)/(EOG)	xlate file record
		EOF	
		Error	Error info

RECORD HEADER BYTES:

Byte 0	EOR	ON = EOR	
Byte 1	EOG	ON = EOG	ON ==> 1
Byte 2	EOF	ON = EOF	OFF ==> 0
Byte 3	Direction	OFF = host-to-PC	
Byte 4	Rec Level	OFF = file-level command	
Byte 5	Key Only	OFF = entire record	
Byte 6	Indicator	1-255 = various codes	
Byte 7-8	Length	0-514 = length of pipe message after	

Figure 7. Information That Flows Between the HCP and the Translation Application

First Message

When a command comes from the host for a transfer of a PSS format file that uses the translation application, the first message that the HCP sends out consists of the record header and the file name,

followed by a CR/LF. The indicator byte in the record header contains an ASCII “O” if the file is the old PSS format and contains an “N” if it is the new PSS format.

For example, if the command is a LOAD of the file named ODATAF, which is the old PSS format, the entire message looks like this:

1	1	0	0	0	0	'O'	0	10	'E'	'A'	'L'	'D'	'A'	'T'	'A'	'F'	CR	LF
---	---	---	---	---	---	-----	---	----	-----	-----	-----	-----	-----	-----	-----	-----	----	----

If the command is a READ KEYED RECORD from the file TEST, which is in the new PSS format, the entire message looks like this:

1	1	0	1	1	1	'N'	0	10	'E'	'A'	'L'	'T'	'E'	'S'	'T'	'F'	CR	LF
---	---	---	---	---	---	-----	---	----	-----	-----	-----	-----	-----	-----	-----	-----	----	----

After the translation application receives this first message from the HCP, the HCP expects a message back with format information about the file and records as they exist on the store controller and at the host. This data is sent with a record header, the format information, and finally with the CR/LF sequence. The format information is grouped by primary input and primary output. For instance, if the command is store controller to host, the primary input are the store controller records and the primary output are the host records.

Figure 8 shows the layout of the format information.

Format Information

A	B	C	D	E	F	G	H	I	J	K
---	---	---	---	---	---	---	---	---	---	---

OTHER VARIABLES

A:N/A
B:N/A
C:N/A

PRIMARY INPUT

D:Record type (K/S)
E:Record format (F/V)
F:Record length (2 bytes)
G:Key length (2 bytes)

PRIMARY OUTPUT

H:Record type (K/S)
I:Record format (F/V)
J:Record length (2 bytes)
K:Key length (2 bytes)

'N' = NEW PSS FORMAT
'O' = OLD PSS FORMAT
'K' = KEYED FILE
'S' = SEQUENTIAL FILE
'F' = FIXED RECORDS
'V' = VARIABLE RECORDS

Figure 8. Layout of Format Information for the First Message

Error Handling

If the HCP discovers an error with either the incoming or the outgoing data, it terminates the translation application program. This termination automatically closes both of the pipes. When the next command comes from the host that requires translation, the starting process described in “First Message” on page 127 occurs to get the translation application loaded and running again.

If the translation application discovers an error, it takes one of the following actions:

- It sends the error in the next record header. This error causes the HCP to terminate the translation application.
- The translation application terminates itself.

If the translation application does terminate with an error, the HCP learns of the situation on the next read from the pipe and notifies the host.

The indicator byte in a record header from the translation application is presently used only to signal that some error has occurred (if it is anything except zero). Any value other than zero notifies the HCP that an error has occurred. The HCP then terminates the translation application if it has not terminated itself.

The following list describes error conditions that the HCP can detect. These error conditions cause the HCP to stop the translation application. These conditions are in addition to errors that the translation application itself detects and reports to the HCP:

- If the HCP requires a message from the translation application that is not received before a fixed time elapses.
- If the file is new data format, and the translation application specifies any file format other than variable sequential.
- If the file type returned is keyed at the store controller and the format is not fixed.
- If the file type returned is keyed at the store controller and the key length at the store controller does not match that returned by file services or if the key length is zero.
- If the file type returned is keyed at the host and the key length is zero.
- If the file format returned for either the store controller or the host is fixed and the respective record length is zero.
- If a record-level command is being processed and the record length returned by the translation application is not the same as that returned by file services.
- If old PSS data is being processed and a record is returned from the translation application which is to go to the host and which is greater than 256 bytes long or which crosses between two pipe messages.
- If the HCP sent the EOF indication to the translation application on a record-level command and the translation application returns anything other than the EOF indication.
- If the translation application returns the EOF indication and the HCP has not sent the EOF indication yet.
- If no data records have been sent from the HCP before the EOF indication is sent, no data records can be returned by the translation application, only the EOF indication.

Sample Flows From the HCP

Table 41 on page 130 shows all of the commands currently supported by the HCP with ADCS. The table shows the settings for header bytes 3, 4, and 5 for the various types of data that can come across the pipe from the HCP.

Notes:

1. **SC** represents the store controller.
2. OFF indicates 0.
3. ON indicates 1.
4. Record can contain 'Error 0720FF ...' if a read error is encountered on disk.

Table 41. Interface Dialogues

Command	Data Area	(Byte 3) Direction	(Byte 4) Rec. Level	(Byte 5) Key Only
DUMP	file name record record : : EOF	SC-Host SC-Host SC-Host	OFF	
LOAD	file name record record : : EOF	Host-SC Host-SC Host-SC	OFF	
CREATE	file name EOF	Host-SC	OFF	
ADD KEYED	file name record record : : EOF	Host-SC Host-SC Host-SC	ON	
REPL KEYED	file name record record : : EOF	Host-SC Host-SC Host-SC	ON	
READ KEYED	file name key record EOF	SC-Host Host-SC SC-Host	ON	ON ON
DEL KEYED	file name key key : : EOF	Host-SC Host-SC Host-SC	ON	ON ON ON

Chapter 7. Using the Remote Command Processor

This chapter introduces the Remote Command Processor (RCP) and explains its usage.

Introduction to RCP

The Remote Command Processor (RCP) is a 4690 application that enables you to perform the following functions:

- Run the 4690 system applications on a remote 4690 system from a central host site
- Initiate applications on the store controller
- Run any user application on a remote 4690 system from a central host site
- Initiate an application on any node if using the Multiple Controller Feature

For example, if a local store is experiencing system problems, the host site located in a remote city needs a system log report from the store to analyze the problem. The RCP application is used to format the report. When the report is formatted, it can be retrieved by the host processor in the remote city for printing and analysis.

The exchange of information between the host site and RCP happens as follows:

1. The host site creates and transmits a SELECTION file and COMMAND file (or files) to the remote store controller.
2. The host site sends the command to start RCP on the remote store controller.
3. RCP reads the SELECTION file to find the name of the appropriate COMMAND file.
4. RCP finds and reads the COMMAND file and begins formatting the system log.
5. The host site retrieves the formatted report.

To use RCP, you must first create two files: an RCP SELECTION file and an RCP COMMAND file. You create both files at the host site and transmit the files to a store controller at a remote site. RCP creates or appends to a third file, the RCP STATUS file, when RCP executes.

Although RCP is a background application, it is not automatically started when the system IPLs. You must start the application each time you want to use it. See “Starting RCP” on page 172 for more information about starting RCP.

Definitions

In this discussion, the *host store controller* refers to a 4690 store controller located at the host site.

Note: A host store controller is needed only if you are applying software maintenance from the host site.

The *remote store controller* refers to a 4690 store controller located at a non-host site. The *host processor* refers to the mainframe computer located at the host site.

The RCP Selection File

The RCP SELECTION file resides in the ADX_IDT1 subdirectory on the hard disk drive of the remote store controller. The RCP SELECTION file, ADXCSHCF.DAT, contains the name of the RCP COMMAND file to be used.

The RCP SELECTION file might contain one or more COMMAND file names. RCP opens the COMMAND file specified and finds the commands to execute. The COMMAND file name can have any of the following levels associated with it:

File name

RCP assumes the file resides in ADX_IDT1 and has an extension of DAT.

`ADX_IDT1:filename.DAT`

File name and extension

RCP assumes the file resides in ADX_IDT1.

`ADX_IDT1:filename.ext`

Subdirectory, file name and extension

RCP takes the information as specified to gain access to the COMMAND file. For LAN users, a two-character node ID (such as DD) might prefix the other parameters.

`DD::fn.ext`

or

`DD::subdirectory:fn.ext`

In a LAN environment, the distribution attributes of the RCP SELECTION file must be Compound, Distribute On Close. The SELECTION file can contain one or more COMMAND file names, but each file name must be on a separate line. In addition, each line must be ended by ASCII CR/LF characters.

Because you can have several RCP COMMAND files, you must edit the RCP SELECTION file each time you use it to ensure that the file contains the appropriate COMMAND file name. For example, if you want to use ASM and that RCP COMMAND file is named ASM.DAT, you would edit the RCP SELECTION file to ensure that it contained the file name ASM.DAT.

After building the RCP SELECTION file, transmit the file to the remote store controller using the HCP LOAD FILE command or using the NetView DM LOAD command. When you are ready to initiate an RCP function, start RCP as described in “Starting RCP” on page 172.

The RCP COMMAND File

The RCP COMMAND file contains commands and parameters for applications. RCP starts the applications on the 4690 system that has RCP running. You must transmit the file created at the host site from the host to your defined subdirectory on the remote store controller. You must send it to the subdirectory that has been named or assumed in the SELECTION file.

The COMMAND file can be any name you choose. If you are using the HCP on the 4690 controller, you should use the file naming standards for the HCP. When using NetView DM, use the guidelines from the operating system. The COMMAND file name must match the file name in the SELECTION file. In a LAN environment, the COMMAND file must be on the store controller that is physically connected to the host line. If the COMMAND file is sent to a 4690 store controller on a LAN, the COMMAND file must be a local file.

The file can contain one or more commands, but each command must be on a separate line. Each line must be ended by ASCII CR/LF characters. Each command can be prefixed by a two-character node ID, subdirectory name, or both. If there is not a prefix to the command, RCP assumes the command resides in the ADX_SPGM subdirectory. Therefore, the command is started on the store controller that is running RCP. The parameters following the commands have the same restrictions as the ADXSTART command in BASIC. The parameters must be no more than 46 characters in length. Any parameters over 46 characters are ignored.

You might want to build several RCP COMMAND files to prepare for different situations. After building an RCP COMMAND file, transmit the file to the remote store controller using the HCP LOAD FILE command or NetView DM LOAD command.

The following example shows commands and parameters that can be included in a COMMAND file of a dump formatter command, followed by a system log formatter command:


```
ADXCSL0L Y 1
CC::ADX_SPGM:ADXCNS0L N 7 02/10/90 03:00 02/10/90 23:50 * * *
ADX_UPGM:USRPROG
```

Note: Certain commands cause the store controller to IPL. If the COMMAND file includes these commands, the commands following them are not executed when RCP processes the COMMAND file.

Command Files on LAN Systems

You can request that a program be started on a specific node in a LAN system by prefixing the application name in the COMMAND file with a two-character node ID followed by two colons (::). This function lets you start applications on any node in the LAN system from the store controller connected to the host.

If you need to run an application on every store controller in the LAN, you can use one command prefixed by two asterisks and two colons (**:). The asterisks tell RCP to start the application on every store controller in the LAN.

The following example COMMAND file starts ASM on store controller JJ:

```
JJ::ADXCST0L N 1 S
```

The following example starts the module-level report formatter on all controllers in a LAN. A single output file named ADX_SDT1:ADXCSSDF.DAT exists on the master controller. This file contains all the reports from each store controller.

```
**::ADXCSS0L N 3 S
```

User-Created Output Files

In a LAN environment, output files created by applications started by RCP must be defined to RCP. Defining the files enables RCP to move files to the master store controller where you can retrieve them by the host. For the Toshiba system applications, you define these output files to RCP by the ADX_SPGM:ADXCESHOF.DAT file. This file lists the 4690 applications and the output files associated with each application.

A similar file is available that identifies output files from user applications. The name of this file is ADX_SDT1:ADXCESHUF.DAT. This file should be used only when the output file from a user application is to be placed on the master store controller after the application has run.

You can use any editor to build the ADX_SDT1:ADXCESHUF.DAT file. The user output file consists of two types of entries:

Application Identification Entry

The *applname* part of this entry is 8 characters or less.

The extension associated with the application name must not be included on this entry.

-applname

Output File Identification Entry

This entry follows the application identification entry. You can have as many output file identification entries following the application identification entry as needed to identify all of the output files. The format of the entry is:

subdir:fn.ext

The following is an example of a user output definition file ADXCESHUF.DAT:

```
-USERAPPL
  ADX_UDT1:UAOUT1.DAT
  ADX_UDT4:UAOUT2.DAT
-UAPPL2
  ADX_IDT1:SALESRPT.DAT
```

For this example, whenever you run USERAPPL on a LAN system on any store controller other than the master, RCP copies ADX_UDT1:UAOUT1.DAT and ADX_UDT4:UAOUT2.DAT to the master store controller into the same subdirectories from where the files came. Whenever you run UAPPL2, RCP copies ADX_IDT1:SALESRPT.DAT to the master store controller into the ADX_IDT1 subdirectory.

When the user output file exists, it must be a Compound file, which means it exists on every store controller in a LAN environment. The user output name file is not used in a non-LAN environment.

Applying Maintenance on a LAN System through RCP

When applying maintenance on a LAN system through RCP, RCP reports a system status for the maintenance activity. This system status reports the most severe error found on any store controller on the LAN. RCP gathers status from each node in a LAN system and logs that status in the RCP STATUS file ADXCShSF.DAT on the master store controller during the IPL processing of RCP. If you are using NetView DM 1.2 or a later level at the host and you are starting RCP with the NetView DM ASYNCHRONOUS CLIST, the status reported to NetView DM using the QUERY command is the LAN status.

The RCP STATUS File

When you run RCP and a 4690 Store System application or a user application that supports the STATUS file, the application creates or updates the STATUS file. The RCP STATUS file ,ADX_SDT1:ADXCShSF.DAT, reports the results of the application. The file contains the commands and parameters that were attempted when the requested program was executed. The file tells you if the formatter you used is currently running, has failed, or has completed.

The STATUS file reports the results of all applications started. The 4690 application logs messages into the RCP STATUS file. You can also write user applications that log messages in the STATUS file.

Table 42 explains some of the messages that are logged.

Table 42. RCP STATUS File Entries

Message	Explanation
dd/dd/dd tt:tt:tt The Remote Command Processor ended.	Includes date and time followed by a message indicating that RCP has ended.
dd/dd/dd tt:tt:tt The Remote Command Processor was started in store xxxx.	Includes the date and time followed by a message indicating that RCP has started.
The current node identifier is nn.	This message is written only when RCP is running on a LAN system. <i>nn</i> is the node ID where RCP is running. This message tells you on which node the command following is executed. The message is not displayed in a non-LAN environment.
dd/dd/dd tt:tt:tt The Remote Command Processor was started in store xxxx due to a store controller IPL.	This message is written when RCP was started by IPL of the store controller and the IPL parameter passed to RCP.
tt:tt:tt Preparing to start a command. tt:tt:tt Command execution is complete.	These messages are written when RCP is attempting to start a command on another node in a LAN environment. You can use these messages as a debugging aid if other expected messages are not written in the STATUS file, especially when starting an application. RCP writes the first message just before it attempts to start a command. RCP writes the second message when the it recognizes the command is no longer running.
tt:tt:tt RUNNING command parameters tt:tt:tt SUCCESS command parameters tt:tt:tt FAILURE command parameters tt:tt:tt PARTIAL command parameters	These messages are written as a result of the 4690 applications running. The messages tell you the status of the command execution.

Table 42. RCP STATUS File Entries (continued)

Message	Explanation
All entries in COMMAND file <i>fn.ext</i> have previously been executed.	This message is written in the RCP STATUS file when RCP has started an IPL of the store controller and determined that all the commands in the COMMAND file have been executed.
Unable to copy ADXLXnnN::ffffffff to the master node. Return code = xxxxxxxx	RCP could not move a file associated with an application that has just completed to the master. Such files are defined in the ADXCShOF.DAT file for Toshiba applications, and in the ADXCShUF.DAT file for user applications. This causes RCP to end with a return code of xFFFFFFFF if other errors have not occurred. Check the RCP STATUS file to determine if the return code for the file move indicates an actual error for this RCP session. For example, if a System Log scan was just ran to create a formatted output file (ADXCsnRF.DAT), and an unformatted output file (ADXCsnUF.DAT) was not created, the message would indicate that ADXCsnUF.DAT could not be moved (RC = 80204010). This return code indicates a file not found condition. In this instance this is not an error because the unformatted output file was not created.
The return code from the previous command is xxxxxxxx.	This message is written when RCP has determined that the command started by RCP is finished. RCP provides the return code in hexadecimal.
The final return code is xxxxxxxx.	This message written when RCP has started all the commands requested in the COMMAND files, and provides the lowest value return code from all the commands in hexadecimal.

The STATUS file has at least two entries for each application that runs. The first entry contains the following elements:

- Time the application was started
- Processing status of the application
- Application name and parameters

The second entry indicates the application completion status, which can be one of the following:

Success	Function completed without problems.
Failure	Function did not complete successfully.
Partial	Only a portion of the function was performed or none of the function was completed, and the incorrect parameters are identified.

The time, the name, and the parameters of the application are listed after the completion status. The next line identifies the return code. Other processing messages might appear between the first and last entries.

You can reset the STATUS file in two ways:

- If you are using NetView DM to start RCP, you can specify the STATUS file reset parameter for the RCP command. NetView DM has the ability to pass parameters using the INITIATECLIST function.
- If you are using ADCS to start RCP, you can specify the STATUS file reset parameter on the command that you are using. Specifying the reset parameter deletes the current file and creates a new one.

An example of a STATUS file follows:

01/05/90 10:05:39 The Remote Command Processor started in store 205.

RUNNING 10:05:45 ADXCS30L N 1 ADX_SMNT:

SUCCESS 10:06:18 ADXCS30L N 1 ADX_SMNT:

The return code for the previous command is 00000000.

01/05/90 10:06:27 The Remote Command Processor ended.

The final return code is 00000000.

RCP Commands

This section explains RCP commands and their parameters. You can include these commands in your RCP COMMAND file as needed. All mandatory parameters for a command are listed. Optional parameters are enclosed in brackets.

Use the following command to start RCP from the host.

Format

ADXCSH0L *i* [*x*] [*o*]

Where:

- i** = The STATUS file reset indicator:
 - Y** = Reset the RCP STATUS file before executing any command.
 - N** = Do not reset the RCP STATUS file.
- x** = The IPL indicator:
 - IPL** = RCP started as a result of an IPL and special processing must be performed.
- o** = The options associated with RCP:
 - 2** = The alternate STATUS file ADX_SDT1:ADXCSHAF.DAT is to be used for messages generated by RCP. You can start RCP from an application using the ADXSTART support. A separate STATUS file from the remote RCP applications and the local RCP applications is available on the remote command processor. When you use the -2 parameter, RCP creates the file ADX_SDT1:ADXCSHAF.DAT when RCP starts.

You can start RCP from the host using either ADCS or NetView DM. If starting RCP with NetView DM, you can specify the status reset parameter. If starting RCP with ADCS, you cannot specify the status reset parameter.

If you use NetView DM, it is recommended that you use the STATUS file reset parameter on the RCP command to reset the STATUS file. Set the status reset indicator to **N** on commands that are in the COMMAND file. By entering **Y** for the STATUS file reset parameter in the COMMAND file, you erase the first record that RCP writes into the STATUS files. The last record is logged telling you when RCP ended.

For more information on starting RCP, see “Starting RCP” on page 172.

Re-IPL Command

This command re-IPLs a store controller.

Format

ADXCS20L *i n*

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset the RCP STATUS file.

n = re-IPL request:

11 = Re-IPL all terminals.

13 = Reload store controller executing these request.

14 = Reload all store controllers on LAN system.

16 = Cold boot store controller executing this RCP request (for Enhanced mode). For Classic controllers, this is the same as option 13 above.

Example

ADXCS20L Y 14

This command resets all store controllers on the LAN if the command is executed at the master store controller.

Remote STC Command

This command requests that Remote STC be run in a single terminal from the controller.

Note: Running remote STC on all terminals exists through RCP.

Format

ADXCS20L *i n t*

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset the RCP STATUS file.

n = Remote STC Request on:

12 = all terminals

15 = specific terminals

t = Terminal number - applicable only when using **15** (specific terminals).

Example

ADXCS20L Y 15 34

This command requests that Remote STC be run in terminal 34.

Dump Formatter Command

This command formats an existing dump into a report file.

Note: The Dump Formatter command does not cause or start a dump. It only formats an already existing dump into a report file.

Format
ADXCSL0L *i n*

Where:

i = STATUS file reset indicator:
 Y = Reset the RCP STATUS file before the command executes.
 N = Do not reset the RCP STATUS file.
n = Type of dump to be formatted:
 1 = Terminal dump.
 2 = Store controller dump.

Example

ADXCSL0L Y 2

This command resets the RCP STATUS file and requests that a store controller dump be formatted in the report file named ADX_SDT1:ADXCSLRF.DAT.

Trace Commands

A trace is a collection of data exchanged between two devices for a specific length of time. You can run a trace from the store controller hard disk drive, host line, and device channel. The demands that are placed on the hard disk drive, controller processor, and device channel affect performance data. You can collect this data by issuing trace commands through RCP. The COMMAND file contains the commands and parameters that initiate the Start/Stop Trace. The format of the commands and parameters are described in the following commands.

Note: System traces and performance traces are mutually exclusive.

Start/Stop Trace Command

When the system trace or performance is started, the data is collected in the file ADXCSOHF.DAT. After you stop performance or the system trace, you can format the collected data through the trace formatter using RCP.

Start Performance Command

This command starts the performance data collection for either a controller or a terminal. The data is collected for a specified time. You can monitor terminals for up to 60 minutes, and you can monitor controllers for up to 24 hours.

Format:
ADXCSQ0L *i 1 t d*

Where:

i = STATUS file reset indicator:
 Y = Reset the RCP STATUS file before the command executes.
 N = Do not reset the RCP STATUS file.
t = Monitoring time for terminals in minutes (10 - 60). Monitoring time for controllers in hours (0 - 24).
 t is a five-character starting time indicated as *HH:MM*, where:

HH = Time in hours (0 - 24)
: = Colon (:) or period (.)
MM = Time in minutes (0 - 59)
d = Device identification:
1–999 Terminal number in decimal
***** Controller

Note: You can use the *HH:MM* format only with controller monitoring. The device identification must be an asterisk (*). The maximum time you can set is 24:00, and the time must be in the five-character format. For example, 1:30 a.m. must be entered as 01:30 or 01.30.

Example 1

```
ADXCSQ0L Y 1 10 *
```

This command resets the STATUS file and requests performance monitoring for the store controller for 10 minutes.

Example 2

```
ADXCSQ0L Y 1 30 23
```

This command resets the STATUS file and requests performance monitoring for terminal number 23 for 30 minutes.

Example 3

```
ADXCSQ0L Y 1 24:00 *
```

This command resets the STATUS file and requests performance monitoring for the store controller for 24 hours.

Format Performance Data Command

This command formats the data received from the Start Performance Data Command and stores the data in the ADXCSPRF.DAT file in the ADX_SDT1 subdirectory.

Format

```
ADXCSPOL i b o s t e t
```

Where:

i = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.
b Available reports:
1 = All
2 = Disk
3 = CPU
4 = Loop
5 = Device channel

- o* Report style:
 - 1** = Graph
 - 2** = Raw utilizations
- st* Five-character starting time in an *HH:MM* format. The default is to report all the data collected.
 - HH** = Time in hours (9-24)
 - :** = Colon (:) or period (.)
 - MM** = Time in minutes (0-59)
- et* Five-character ending time in an *HH:MM* format. This parameter is required if you select a starting time.

Note: The first three parameters are required. *et* and *st* are only for reporting data that has been collected for more than one hour.

Example 1

```
ADXCSP0L Y 1 2
```

This command resets the STATUS file and requests that the data for all devices for the entire collection period be in a raw utilization format.

Example 2

```
ADXCSP0L Y 3 1 02:30 14:30
```

This command resets the STATUS file and requests that the data collected for the CPU from 2:30 a.m. until 2:30 p.m. be in a graphical output.

Start Trace Command for Loop, Hard Disk Drive, and Host Line

Note: Store Loop support was removed in 4690 OS V6R3.

This command collects trace data for loop, hard disk drive, and host line devices.

Format:

```
ADXCSQ0L i 2 p w
```

Where:

- i* = STATUS file reset indicator:
 - Y** = Reset the RCP STATUS file before the command executes.
 - N** = Do not reset the RCP STATUS file.
- p* = Trace type:
 - 1** = Loop
 - 2** = Disk
 - 3** = Host line
- w* = Wrap character:
 - Y** = Wrap when trace full.
 - N** = No wrap; stop tracing when trace is full.

Example

```
ADXCSQ0L Y 2 2 Y
```

This command resets the STATUS file and collects trace data for the hard disk drive, wrapping when the trace is full.

Start Trace Command for Device Channel

This command collects trace data for the device channel.

Format:

```
ADXCSQ0L i 2 4 d w
```

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset the RCP STATUS file.

d = Device identification:

1–999 Terminal number in decimal

w = Wrap character:

Y = Wrap when trace full.

N = No wrap; stop tracing when trace is full.

Example

```
ADXCSQ0L Y 2 4 15 N
```

This command resets the STATUS file and starts a trace for the device channel for terminal 15, stopping the trace when the trace is full.

Start Trace Command to Start All Traces

This command starts the collection of all trace data.

Format

```
ADXCSQ0L i 2 * d w
```

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset STATUS file.

d = Device identification for device control

1–999 Terminal number in decimal

w = Wrap character:

- Y =** Wrap when trace is full.
N = No wrap; stop tracing when trace is full.

Example

```
ADXCSQ0L N 2 * 777 Y
```

This command starts a trace for the device channel for terminal number 777 and wraps when the trace is full. This example does not reset the STATUS file.

Start Ethernet Trace Command

This command starts an Ethernet trace.

Format

```
ADXCSQ0L i 2 6 f w
```

Where:

i = STATUS file reset indicator:

- Y =** Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.

f = Change configuration option parameter:

- 1 =** Any Service Access Point (SAP)
2 = Terminal Controller Communications (TCC)

Note: See the following section for tracing Terminal Controller Communications (TCC).

3 = SNA

4 = NetBIOS

5 = TCP/IP

6 = Remote Program Load (RPL)

7 = Specific SAP value

Note: See the following section for tracing a Service Access Point (SAP).

w = Wrap character:

- Y =** Wrap when trace is full.
N = Stop tracing when the trace is full.

Ethernet Trace (TCC):

Format

```
ADXCSQ0L i 2 6 2 d w
```

This command starts an Ethernet trace for an Ethernet TCC.

Where:

i = STATUS file reset indicator:

- Y =** Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.

d = Device identification:

1–999 = Terminal Number in decimal

w = Wrap character:

- Y =** Wrap when trace is full.

N = Stop tracing when the trace is full.

Ethernet Trace Service Access Point (SAP): This command starts an Ethernet trace for a Service Access Point.

Format

ADXCSQ0L i 2 6 7 s w

Where:

i = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.

s = Service Access Point (2 hex digits):
EB = TCC
04 = SNA
F0 = NetBIOS
AA = TCP IP
FC = RPL
F8 = RPL
xx = any other specific SAP value.

w = Wrap character:
Y = Wrap when trace is full.
N = Stop tracing when the trace is full.

Example

ADXCSQ0L Y 2 6 7 04 N

This example has a Service Access Point value of 04 and the trace stops when it is full.

If starting the trace generates any messages, they can be viewed by checking the contents of the STATUS file ADX_SDTI:ADXCSHSF.DAT.

Stop Performance Command

This command stops the collection of performance data.

Format:

ADXCSQ0L i 3

Where:

i = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.

Stop Trace Command

This command stops a system trace.

Format

ADXCSQ0L i 4

Where:

- i* = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.

Trace Formatter Command

This command formats system trace data into a report file.

Note: This command does not start a trace. It only formats existing trace data into a report file.

Format

ADXCSM0L *i t d*: [*f* [*a*]]

Where:

- i* = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.
- t* = Type of trace to be formatted:
1 = All
2 = Loop
3 = Disk
4 = Host
5 = Device channel
7 = Ethernet
- d* = Optional. Disk ID when the type of trace to be formatted is a disk. If this parameter is not specified, the output defaults to the C: disk drive.
- Note:** An * can be specified to format all disk trace entries.
- f* = Type of output file to be created. If this parameter is not specified, the output defaults to formatted output. The formatted output file contains only one report. This parameter is optional.
- F** = Formatted system trace report file (ADX_SDT1:ADXCSMTF.DAT), or Performance monitor report file (ADX_SDT1:ADXCSPRF).
- U** = Unformatted trace report file (ADX_SDT1:ADXCSMUF.DAT).
- a* = Action to be performed on the trace file. If this parameter is not specified, the output file contains only one report. This parameter is optional.
- A** = Append the output trace report file to the current trace output file.

Example

ADXCSM0L Y 3 D: F

This command resets the RCP STATUS file and requests the D: disk trace entries to be formatted to the report file named ADX_SDT1:ADXCSMTF.DAT.

File Management Command for Trace Output Data Files

Two output files locate system trace data:

- The formatted report file (ADX_SDT1:ADXCSMTF.DAT)
- The file containing unformatted trace entries (ADX_SDT1:ADXCSMUF.DAT)

Two questions need to be answered about the output files to support trace formatting:

1. Do you want to append data to the end of the current output files?

2. Do you want to erase data in the output files before writing to them?

To save the current data and append new data into the current output files, do not use the file management command. To erase the data from either of the current output files, use the file management command for trace output files.

The file management command enables you to identify the output file and the action to take on the file.

Format

`ADXCSM0L i 8 f o`

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset the RCP STATUS file.

f = Format of file:

F = Formatted system trace report file (ADX_SDT1:ADXCSMTF.DAT).

U = Unformatted file (ADX_SDT1:ADXCSMUF.DAT).

o = Current data status:

A = Append

B = Erase

Note: The command must have the *p*, *f*, *o* parameters or it is treated as a trace formatter command attempt.

Examples of Trace Formatter Command

This section contains several examples. The examples show how you can tailor the trace formatter command for the needs of your enterprise.

Example 1

Command	Explanation
ADXCSM0L Y 8 U B	Erases the unformatted output file and writes an empty header record to the output file.
ADXCSM0L N 2 U	Copies any loop trace data in the ADXCISOHF.DAT file into the ADX_SDT1:ADXCSMU.F.DAT file.
ADXCSM0L N 3 U	Writes the disk trace data to the unformatted output file (ADX_SDT1:ADXCSMU.F.DAT).

After the disk trace data is written to the unformatted output file, the header record is updated. The header record in the unformatted output allows you to format the unformatted data file. Typically, the host retrieves this file and transmits it to a master 4690 store controller. After erasing existing trace data files, the file can be renamed to ADXCISOHF.DAT, and you can use the screen interface to format the trace data collected in this file.

The unformatted file has no size restrictions. The traces are appended and can be processed later. If more than one type of trace is in the unformatted file, you can process all or selected traces. For example, you can process all disk trace entries in the order they were added to the file.

Example 2

Command	Explanation
ADXCSM0L Y 8 F B	Erases the data in the formatted output file ADX_SDT1:ADXCSMTF.DAT.
ADXCSM0L N 2 F	Formats any loop trace data and places it in the formatted output file.
ADXCSM0L N 3 F	Formats any disk trace data and appends it to the formatted output file.

System Log Formatter Command

This command enables you to format the system log. You specify which errors or events you want reported and the date and time interval.

Format

ADXCSN0L *i b sd st ed et c t s [r]*

Where:

- i* = STATUS file reset indicator:
 - Y** = Reset the RCP STATUS file before the command executes.
 - N** = Do not reset the RCP STATUS file.
- b* = The log you choose to be reported and the format type:
 - Short formats:
 - 1** = Store controller hardware errors
 - 2** = Terminal hardware errors
 - 3** = Terminal events
 - 4** = Store controller events
 - 5** = System events
 - 6** = Application events
 - 7** = Format all sections 1 through 6
 - Long formats:

A = Store controller hardware errors
B = Terminal hardware errors
C = Terminal events
D = Store controller events
E = System events
F = Application events
G = Format all sections A through F

sd = 8-character starting date in the format *mm/dd/yy*
mm = Month (01 - 12)
dd = Day of the month (01 - 31)
yy = Year (00 - 99):
 86 to 99 = 1986 to 1999
 00 to 85 = 2000 to 2085

st = 5-character starting time in the format *HH:MM*:
HH = Hour (00 - 23)
MM = Minute of the hour (00 - 59)

ed = Ending date, in the format *mm/dd/yy*
et = Ending time, in the format *HH:MM*

c = Store controller ID:
CC - ZZ = Store controller ID
*** =** All store controller IDs. This value is valid if the section of the system log being reported is store controller related.

If the section of the system log is terminal-related, this field is ignored.

t = Terminal ID:
1 to 999 = Terminal IDs in decimal
*** =** All terminal ID. This value is valid if the section of the system log being reported is terminal-related.

If the section of the system log is store controller-related, this field is ignored.

s = Source number:
001-127 = Source numbers
*** =** All source numbers

See the *4690 OS: Messages Guide* for source numbers and their descriptions.

r = Type of output to be acted on. If this parameter is not coded, the formatted output file contains only this report. This is an optional parameter.
1 = Format report into new output file (ADX_SDT1:ADXCSNRF.DAT).
2 = Append formatted report to current output file.
3 = Put unformatted data into new output file (ADX_SDT1:ADXCSNUF.DAT).
4 = Append unformatted report to current output file.

Example

```
ADXCSN0L N 4 11/12/86 14:00 11/17/86 14:00 * * *
```

This command requests a system log report for store controller events from all sources. The example requests the report to cover events beginning at 2:00 p.m. on 11/12/86 (November 12, 1986) and ending at 2:00 p.m. on 11/17/86 (November 17, 1986). The report is formatted into the file named ADX_SDT1:ADXCSNRF.DAT.

File Management Command for System Log

You can create two types of output files:

1. Formatted Report File (ADX_SDT1:ADXCSNRF.DAT)
2. Unformatted Report File (ADX_SDT1:ADXCSNUF.DAT)

A system log formatter command can create the output files by appending data or by erasing data. If you want to append data to the end of an output file, the file management command is optional. The formatting command defaults to append data to the existing output file.

If you want to erase the current data in an output file before issuing the formatting command, you should run the file management command. The file management command enables you to identify the output file and the action to take with the file.

If you do not use the file management command to clear the output files, options of the system log formatter command allow you to perform the same task.

Format
`ADXCSN0L i 8 r z`

Where:

i = Status reset indicator:
 Y = Reset the RCP STATUS file before the command executes.
 N = Do not reset the RCP STATUS file.
r = Output file to be affected:
 F = Formatted output file (ADX_SDT1:ADXCSNRF.DAT)
 U = Unformatted data output file (ADX_SDT1:ADXCSNUF.DAT)
z = Action for the output file:
 A = Append data to output file (do not reset the output file)
 B = Reset the output file (erase existing data in file)

Examples of System Log Formatter Command

This section contains several examples. The examples show how you can tailor the system log formatter command for the needs of your enterprise.

Example 1

```
ADXCSN0L Y 8 F B
ADXCSN0L N 1 02/24/87 12:00 02/25/87 12:00 * * * 2
ADXCSN0L N 3 02/24/87 12:00 02/25/87 12:00 * * * 2
ADXCSN0L N 5 02/24/87 12:00 02/25/87 12:00 * * * 2
```

This example creates a formatter report file that contains only the data from the store controller hardware log, terminal event log, and the system event log. The three reports in this example are appended to the existing formatted report file.

Example 2

```
ADXCSN0L Y 8 U A
ADXCSN0L N 1 02/24/87 12:00 02/25/87 12:00 * * * 4
ADXCSN0L N 3 02/24/87 12:00 02/25/87 12:00 * * * 4
ADXCSN0L N 5 02/24/87 12:00 02/25/87 12:00 * * * 4
```

This example keeps the data previously contained in the unformatted output file. The last three commands append the unformatted data from the store controller hardware log, the terminal event log, and the system event log to the existing unformatted output file.

Example 3

```
ADXCSD0L N 1 02/24/87 12:00 02/25/87 12:00 * * * 2
ADXCSD0L N 3 02/24/87 12:00 02/25/87 12:00 * * * 2
ADXCSD0L N 5 02/24/87 12:00 02/25/87 12:00 * * * 2
```

When the file management command is not used, as in this example, the resulting formatted report file contains the data previously in the file. These commands append the reports for the store controller hardware log, terminal event log, and the system event log to the existing formatted output file.

Configuration Activation Command

This command begins activation of the configuration specified by the user-provided parameter.

Format

```
ADXCSD0L i x y
```

Where:

- i* = STATUS file reset indicator:
 - Y** = Reset the RCP STATUS file before the command executes.
 - N** = Do not reset the RCP STATUS file.
- x* = Change configuration option parameter:
 - 4** = Activate configuration. All other values are invalid.
- y* = Activation configuration option parameter:
 - 1** = Terminal configuration
 - 2** = Controller configuration
 - 3** = System configuration

Example

```
ADXCSD0L Y 4 2
```

This example command causes the store controller to attempt activation of the controller configuration files.

Error checking exists to verify the number and the values of the parameters that were passed to the controller. If this activation of the configuration causes any messages, they can be viewed by checking the contents of the STATUS file ADX_SDT1:ADXCSDSF.DAT.

Configuration Data Formatter Command

This command creates reports on the configuration of your terminals, controllers, and system.

When reporting on store controller configuration, the report contains information on the controller that performed the command. To obtain controller configuration information for a specific store controller on a LAN system, you must instruct RCP to run the command on the store controller of interest. For example, you have a two-controller LAN system with nodes CC and DD. Node CC has the host line attached to it. To obtain controller configuration information for node DD, your RCP COMMAND file must contain this command:

```
DD::ADXCSD0L N 3 1
```

Format

```
ADXCSD0L i n s [a]
```

Where:

- i* = STATUS file reset indicator:
 - Y** = Reset the RCP STATUS file before the command executes.
 - N** = Do not reset the RCP STATUS file.
- n* = Type of configuration reported:
 - 1** = All of configuration
 - 2** = Terminal configuration
 - 3** = Controller configuration
 - 4** = System configuration
 - 5** = SurePOS™ 300/700 Series terminal configuration
- s* = Configuration status:
 - 1** = Active configuration
 - 2** = Inactive configuration
- a* = Action to perform on the output file. If this parameter is not coded, the configuration report output file contains only this report.
 - A** = Append report to current output file.

Example

```
ADXCSD0L Y 3 1
ADXCSD0L N 2 2 -A
```

This example creates a formatted report file that contains the data for the store controller and terminal configurations. The command saves the report in the file named ADX_SDT1:ADXCSDTF.

The report will be saved in the file named ADX_SDT1:ADXCSDTF.DAT.

Report Module Level Command

This command creates module-level reports for the operating system and system applications.

Format

```
ADXCSS0L i n [f]p [f]p [f]p
```

Where:

- i* = STATUS file reset indicator:
 - Y** = Reset the RCP STATUS file before the command executes.
 - N** = Do not reset the RCP STATUS file.
- n* = Report type:
 - 1** = Product Summary Report
 - 2** = Complete report with module integrity (checksum)
 - 3** = Complete report without module integrity (no checksum)
 - 4** = Module-level APAR report
 - 5** = APAR search
 - 6** = Installed features report

A *checksum* is an operation performed on each module to ensure that the module has not been changed since it was first placed on the 4690 store controller by ASM.

f = Product being reported. The letter f is the fifth letter of the Product Control File name, ADXC*f* p D.DAT.

Note: If the f is missing, S is assumed to be the fifth letter.
 p = Product being reported. The letter p is the seventh letter of the Product Control File name, ADXC*f* p D.DAT.

Note: The combination of f and p are used to determine the product. For example, ADXCSTQD.DAT reports on the remote operator. ADXCSTSD.DAT reports on the base operating system.

Some examples are:

G = 4680 or 4690 Application Program Product (General Sales, Supermarket, or Chain Drug Application)
H = User exits to 4680 Application Program Products
P = Price Management Feature for the General Sales Application
S = Operating System
Q = Remote Operator
B = 4690 Optionals
@ = DDM

You can include a maximum of 21 installed products. If you want to include products other than those shown, you must create a Product Control File for each product.

Example 1

```
ADXCSSL Y 3 S
```

This command resets the RCP STATUS file and requests a module-level report for the operating system, with no checksum operation to be performed. The command saves the report in the file named ADX_SDT1:ADXCSSDF.DAT.

Audible Alarm Commands

This section describes the commands for issuing audible alarms.

Activate Command for a LAN System

This command activates audible alarms for a LAN system.

Format

```
ADXCSSL  $i$  3  $t$   $c$  [ $s$ ]
```

Where:

i = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.
 t = Length of time the alarm is sounded.
1 - 99 = Number of seconds in decimal.
***** = Sound continuously. If the coded value is not valid, the default of “*” is used.

- c =** Store controller ID. Must be present in LAN system as valid ID or *. No default is assumed.
- s =** System message:
- Y =** Use the configured severity level for the system message display panel and the message number to determine if the message is to be alarmed.
 - N =** Use only the message number to determine if the message is to be alarmed. This value is the default.

The default is used if the coded value is not valid.

Activate Command for a Non-LAN System

This command activates audible alarms for non-LAN systems.

Format

ADXCS10L *i* 3 *t* [*s*]

Where:

- i =** STATUS file reset indicator:
- Y =** Reset the RCP STATUS file before the command executes.
 - N =** Do not reset the RCP STATUS file.
- t =** Length of time the alarm is sounded.
- 1 - 99 =** Number of seconds in decimal.
 - * =** Sound continuously. The asterisk (*) is the default.
- The default is used if the coded value is not valid.
- s =** System message:
- Y =** Use the configured severity level for the system message display panel and the message number to determine if the message is to be alarmed.
 - N =** Use only the message number to determine if the message is to be alarmed. This value is the default.
- The default is used if the coded value is not valid.

Examples of an Audible Alarm Command for a LAN System

Example 1

ADXCS10L Y 3 10 *

In this example, the activate command resets the STATUS file and causes a 10-second alarm for all store controllers in the system for any messages in the alarm message files. The system message severity level is ignored because the optional parameter is not present (defaults selection to -N).

Example 2

ADXCS10L Y 3 * * -Y

In this example, the activate command resets the STATUS file and causes a continuous alarm for all store controllers in the system for any messages meeting the system message severity level because the optional parameter -Y indicates YES for message level selection.

Example 3

```
ADXCS10L Y 3 10 EE -Y
```

In this example, the activate command resets the STATUS file and causes a 10-second alarm for controller ID EE's messages meeting the system message severity level because the optional parameter -Y indicates YES for message level selection. No other controller's audible alarm function is affected.

Examples of an Audible Alarm Command for a Non-LAN System**Example 1**

```
ADXCS10L Y 3 20 -N
```

In this example, the activate command resets the STATUS file, causes a 20-second alarm, and sets the message level selection to NO. The system message severity level is ignored.

Example 2

```
ADXCS10L Y 3 * -Y
```

In this example, the activate command resets the STATUS file, causes a continuous alarm and sets the message level selection -Y to YES. Only messages meeting the system message severity level sound the alarm.

Example 3

```
ADXCS10L N 3 2
```

In this example, the activate command appends to the STATUS file and causes a 2-second alarm for messages in the audible alarm file. The system message severity level is ignored (selection defaulted to -N).

Example 4

```
ADXCS10L Y 3 *
```

In this example, the activate command resets the STATUS file and causes a continuous alarm for all messages in the audible alarm message file. The default for message level (option not present) selection is -N.

Example 5

```
ADXCS10L Y 3 10 DD -Y
```

This example illustrates an incorrect store controller ID. This command resets the STATUS file, causes a 10-second alarm, and activates the audible alarm function in a LAN system store controller with a valid ID of DD. For this example, a store controller DD does not exist, so the command fails.

Deactivate Command

This command deactivates the audible alarm command.

Format

ADXCS10L *i* 4 [*c*]

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset the RCP STATUS file.

c = The store controller ID to deactivate:

CC - ZZ

= Store controller ID.

***** = All store controllers. This optional parameter is valid for multiple store controller systems only.

Example

ADXCS10L Y 4 *

This command resets the RCP STATUS file and deactivates the audible alarm on all store controllers.

Report Audible Alarm Status Command

This command creates a report showing the audible alarm status for store controllers.

Format

ADXCS10L *i* 2 [*c*]

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset the RCP STATUS file.

c = Store controller ID whose audible alarm status is to be reported:

CC - ZZ

= Store controller ID.

***** = All store controllers. This optional parameter is valid for multiple store controller systems only.

Example

ADXCS10L Y 2 *

This command resets the RCP STATUS file and runs the report for all store controllers (the report is in ADX_SDT1:ADXCS1RF.DAT).

Apply Software Maintenance Command

This command applies software maintenance to the operating system or system applications.

See the *4690 OS: User's Guide* for more information on the Apply Software Maintenance (ASM) command.

Format

ADXCST0L *i a[f]p ... a[f]p [TL] [BY] [NI]*

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset the RCP STATUS file.

a = Action to be performed on the product:

1 or TST

= Test

2 or CNL

= Cancel

3 or ACC

= Accept

Note: For this parameter, you can use all numeric characters, all alphabetic characters, or both. For example, you can use 1 to test and CNL to cancel in the same command.

f = Product being activated. The *f* is the fifth letter of the Product Control File name. The Product Control File for a product is found on the product's master installation or Corrective Diskette under the format ADXC*f**p*D.DAT.

Note: If the *f* is missing, *S* is assumed to be the fifth letter. The *S* indicates that the report module level assumes the maintenance is in ADX_SMNT (the system maintenance subdirectory) unless otherwise specified.

p = Product being activated. The *p* is the seventh letter of the Product Control File name. The Product Control File for a product is found on the product's master installation or Corrective Diskette under the format ADXC*f**p*D.DAT.

Note: The combination of *f* and *p* are used to determine the product. For example, ADXCSTQD.DAT reports on the remote operator. ADXCSTSD.DAT reports on the base operating system.

Some examples are:

G = 4680 or 4690 Application Program Product (General Sales, Supermarket, Store Management, or Chain Drug Application)

H = User exits to 4680 Application Program Products

P = Price Management Feature for the General Sales Application

S = Operating System

Q = Remote Operator

B = 4690 Optionals

@ = DDM

If you want to include products other than those shown above, you must create a Product Control File for each one. You cannot perform two actions on the same product in the same parameter list.

TL = Terminal load indicator. You can cause all terminals to be reloaded upon completion of ASM by including the terminal load indicator, TL, in the command. The TL can be in any position after the STATUS file reset indicator.

- BY =** Bypass LAN problems indicator. You can optionally bypass LAN problems detected by ASM by including the bypass indicator, BY, in the command. BY can be in any position after the STATUS file reset indicator.
- NI =** No IPL indicator. You can optionally get ready for the activation of the maintenance by including the no-IPL indicator, NI, in the command. Checksums are verified, maintenance is distributed, and the activation file is built. All that remains to activate the maintenance is a manual IPL of the controller.

Example 1

```
ADXCST0L Y 1S 2G TL
```

This command resets the RCP STATUS file, puts the product indicated by Product Control File ADXCSTSD.DAT into TEST mode, cancels the maintenance for the product named by the Product Control File, ADXCSTGD.DAT, and reloads all terminals.

Example 2

```
ADXCST0L Y TSTS CNLG TL
```

This command resets the RCP STATUS file, puts the product indicated by the Product Control File, ADXCSTSD.DAT, into test mode, cancels the maintenance for the product named by the Product Control File, ADXCSTGD.DAT, and reloads all terminals.

Streaming Tape Drive Utility Commands

See the *4690 OS: User's Guide* for more information about streaming tape drive utility commands.

Backup Command

This command lets you use the streaming tape drive utility to backup files or subdirectories.

Format

```
ADXCSTV0L i 1 s d f e [IW] x
```

Where:

- i =** STATUS file reset indicator:
 - Y =** Reset the RCP STATUS file before the command executes.
 - N =** Do not reset the RCP STATUS file.
- s =** The files to be backed up:
 - 1 =** All files on a drive
 - 2 =** A subdirectory
 - 3 =** A list of files (see the *4690 OS: User's Guide*)
 - 4 =** One file
- d =** The drive or the subdirectory
 - If **s** is 1:
 - 1 =** Drive C:
 - 2 =** Drive D:
 - 3 =** Both drive C: and D:
 - If **s** = 2, 3, or 4, **d** = Subdirectory name.
- f =** The file name (if **s** is 1 or 2, file name must be an asterisk (*)).
- e =** The action to be done to any data currently on the tape in the streaming tape drive:
 - 1 =** Perform erase before backup.

- 2 =** Overwrite from beginning of tape.
- 3 =** Add a new volume after last volume.
- IW =** Include files open for unlocked write access.
- x =** How to label the backup tape:
 - If **e = 1** or **2**, **x** is the label to be put on the tape.
 - If **e = 3**, **x** is an asterisk (*).

Example

```
ADXCSV0L Y 1 2 ADX_SDT1: * 2 REPORTS FOR 6/26/87
```

This command requests:

- Backup subdirectory ADX_SDT1
- Overwrite any existing data on the tape in the streaming tape drive
- Label the tape "REPORTS FOR 6/26/87"

Restore Command

This command lets you use the streaming tape drive utility to restore files in a tape volume, a subdirectory, or a list.

Format

```
ADXCSV0L i 2 s d f v
```

Where:

- i =** STATUS file reset indicator:
 - Y =** Reset the RCP STATUS file before the command executes.
 - N =** Do not reset the RCP STATUS file.
- s =** The files to be restored:
 - 1 =** Restore all files in a tape volume.
 - 2 =** Restore all files in a subdirectory.
 - 3 =** Restore all files named in a list (see the *4690 OS: User's Guide*)
 - 4 =** Restore one file.
- d =** Subdirectory (if **s = 1**, **d** must be an asterisk (*))
- f =** File name (if **s = 1** or **2**, **f** must be an asterisk (*))
- v =** Volume number (* = all volumes on the tape).

Example

```
ADXCSV0L Y 2 1 * * *
```

This command restores all files in all tape volumes (all files on the tape).

List Command

This command produces a list of the files on a tape.

Format

```
ADXCSV0L i 3 s d f v
```

Where:

- i =** STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.
s = Files to be listed:
 1 = All files in a tape volume
 2 = All files in a subdirectory
 3 = All files named in a list of files (see the *4690 OS: User's Guide*)
 4 = One file
d = Subdirectory (if *s* = 1, *d* must be an asterisk (*))
f = File name (if *s* = 1 or 2, *f* must be an asterisk (*))
v = Volume number (* = all volumes on the tape).

Example

```
ADXCSV0L Y 3 4 C:\ADX_IDT1\ ADXUSR01.DAT 1
```

This command requests a list of file ADXUSR01.DAT, which was backed up from the C: drive in subdirectory ADX_IDT1. The ADXUSR01.DAT file resides on the first volume of the tape.

Erase Command

This command erases the tape in the streaming tape drive.

Format

```
ADXCSV0L i 4
```

Where:

i = STATUS file reset indicator:
 Y = Reset the RCP STATUS file before the command executes.
 N = Do not reset the RCP STATUS file.

Example

```
ADXCSV0L Y 4
```

This command resets the RCP STATUS file and erases the tape in the streaming tape drive.

Adjust Tape Tension Command

This command adjusts the tension of the tape in the streaming tape drive.

Format

```
ADXCSV0L i 5
```

Where:

i = STATUS file reset indicator:
 Y = Reset the RCP STATUS file before the command executes.
 N = Do not reset the RCP STATUS file.

Example

```
ADXCSV0L Y 5
```

This command resets the RCP STATUS file and adjusts the tension of the tape.

Optical Drive Utility Commands

See the *4690 OS: User's Guide* for more information about optical drive utility commands.

Backup Command

This command lets you use the optical drive utility to backup files or subdirectories.

Format:

```
ADXCS60L i 1 o w s f r e w x
```

where:

- i** = RCP STATUS file reset indicator:
 - Y** = Reset RCP STATUS file before the command executes.
 - N** = Do not reset RCP STATUS file.
- o** = Logical drive of target optical drive (usually O:).
- w** = Files to be backed up:
 - 1** = Back up all files on a drive.
 - 2** = Back up a subdirectory.
 - 3** = Back up all files named in a selection list file.
 - 4** = Back up one file.
- s** = Source drive or subdirectory:
 - If w = 1,** **s** = Source drive (for example, D:)
 - If w = 3,** **s** = Drive and subdirectory of the selection list file
 - If w = 2 or 4,** **s** = Source drive and subdirectory

Note: For example, C:\ADX_IDT1\. If the drive is not specified, it will default to the C: drive. The node can also be included. For example, ADXLXAAN::C:\ADX_IDT1\.

- f** = File name (if w = 1 or 2, file name must be *)
- r** = Subdirectories within subdirectories:
 - 1** = Back up only the chosen subdirectory.
 - 2** = Back up subdirectories within the chosen subdirectory.
 - *** = Not applicable (if w = 1, 3, or 4, r must be *).
- e** = Action to be done to any data currently on the optical diskette:
 - 1** = Format, then back up to volume 1.
 - 2** = Create new volume, then back up to the new volume.
- w** = Files open for unlocked write access:
 - 1** = Exclude files open for unlocked write access.
 - 2** = Include files open for unlocked write access.
- x** = Description of the backup data

Example

```
ADXC60L Y 1 0: 2 ADX_SDT1: * 1 2 1 REPORTS FOR 6/26/87
```

This example resets the RCP STATUS file, creates a new volume 1, backs up subdirectory C:/ADX_SDT1 to optical drive O:, and labels the disk "REPORTS FOR 6/26/87".

Restore Command

This command lets you use the optical drive utility to restore files in a tape volume, a subdirectory, or a list.

Format:

```
ADXC60L i 2 o w s f r v d
```

where:

i = RCP STATUS file reset indicator:
 Y = Reset RCP STATUS file before the command executes.
 N = Do not reset RCP STATUS file.
o = Logical drive of source optical drive (usually O:)
w = Files to be restored:
 1 = Restore all files in a volume.
 2 = Restore all files in a subdirectory.
 3 = Restore all files named in a selection list file.
 4 = Restore one file.
s = Subdirectory:
 If w = 1, **s = ***
 If w = 3, **s =** Drive and subdirectory of the selection list file
 If w = 2 or 4, **s =** Target drive and subdirectory

Note: The files are always restored to the same subdirectory from which they were backed up. The target drive defaults to the C: drive unless it is specified as part of the *s* parameter. For example, if files were backed up from C:\ADX_IDT1\, you can restore the files to D:\ADX_IDT1 by specifying D:\ADX_IDT1 as the *s* parameter. Node can also be included. For example, ADXLXAAN::C:\ADX_IDT1\.

f = File name (if *w* = 1 or 2, *f* must be *)
r = Subdirectories within subdirectories:
 1 = Restore only the chosen subdirectory.
 2 = Restore subdirectories within the chosen subdirectory.
 ***** = Not applicable (if *w* = 1, 3, or 4, *r* must be *).
v = Volume number (* = all volumes on the optical disk)
d = Target drive:
 If w = 2 or 4, **w = ***
 If w = 1 or 3, **w =** Target drive

Note: For example, C:\ADX_IDT1\. If drive is not specified, it defaults to the C: drive. Node can also be included. For example, ADXLXAAN::C:\ADX_IDT1\.

Example

```
ADXCS60L Y 2 0: 1 * * * * C:
```

This example requests to restore all files in all volumes on optical drive O: to the C: drive.

List Command

This command produces a list of the files on a tape.

Format:

```
ADXCS60L i 3 o w s f r v
```

where:

- i** = RCP STATUS file reset indicator:
 - Y** = Reset RCP STATUS file before the command executes.
 - N** = Do not reset RCP STATUS file.
- o** = Logical drive of optical drive (usually O:)
- w** = Files to be listed:
 - 1** = List all files in a volume.
 - 2** = List all files in a subdirectory.
 - 3** = List all files named in a selection list file.
 - 4** = List one file.
- s** = Subdirectory:
 - If *w* = 1,** *s* = *
 - If *w* = 3,** *s* = Drive and subdirectory of the selection list file
 - If *w* = 2 or 4,** *s* = Subdirectory to list (drive would be ignored)
- f** = File name (if *w* = 1 or 2, *f* must be *)
- r** = Subdirectories within subdirectories:
 - 1** = List only the chosen subdirectory.
 - 2** = List subdirectories within the chosen subdirectory.
 - *** = Not applicable (if *w* = 1, 3, or 4, *r* must be *).
- v** = Volume number (* = all volumes on the optical disk)

Example

```
ADXCS60L Y 3 0: 4 \ADX_IDT1\ ADXUSR01.DAT * 1
```

This example requests a list of file ADXUSR01.DAT, which was backed up from subdirectory ADX_IDT1 and resides on volume 1 of the optical drive O:.

Erase Command

This command erases the tape in the optical drive.

Format:

```
ADXCS60L i 4 o w s f r v
```

where:

- i** = RCP STATUS file reset indicator:
 - Y** = Reset RCP STATUS file before the command executes.

N = Do not reset RCP STATUS file.
o = Logical drive of optical drive (usually O:)
w = Files to be erased:
 1 = Erase all files in a volume.
 2 = Erase all files in a subdirectory.
 3 = Erase all files named in a selection list file.
 4 = Erase one file.
s = Subdirectory:
 If w = 1, **s** = *
 If w = 3, **s** = Drive and subdirectory of the selection list file
 If w = 2 or 4, **s** = Subdirectory to erase (drive would be ignored)
f = File name (if *w* = 1 or 2, *f* must be *)
r = Subdirectories within subdirectories:
 1 = Erase only the chosen subdirectory.
 2 = Erase subdirectories within the chosen subdirectory.
 ***** = Not applicable (if *w* = 1, 3, or 4, *r* must be *).
v = Volume number (* = all volumes on the optical disk)

Example

```
ADXCS60L Y 4 O: 4 C:\ADX_IDT1\ ADXUSR01.DAT * 1
```

This example requests a erase of file ADXUSR01.DAT, which was backed up from subdirectory ADX_IDT1 and resides on volume 1 of the optical drive O:.

Format Command

This command lets you format the diskette in the optical drive.

Format:

```
ADXCS60L i 5 o t
```

where:

i = RCP STATUS file reset indicator:
 Y = Reset RCP STATUS file before the command executes.
 N = Do not reset RCP STATUS file.
o = Logical drive of optical drive (usually O:)
t = Format type:
 1 = Short format
 2 = Long format

Example

```
ADXCS60L Y 5 O: 1
```

This example resets the RCP STATUS file and formats the optical disk.

Load All Terminals Command

This command enables you to load all terminals in your store. After maintenance on applications, terminals should use new modules and files. Terminals must be reloaded to do this. You can use this function to reload all terminals in a remote environment or if you do not have operators working near your store controllers.

Note: The RCP STATUS file, (ADX_SDT1:ADXCSHSF.DAT) contains the completion status of the command. The status does not include the terminal IDs of the terminals that have been reloaded.

Format

ADXCS20L *i n*

Where:

i = STATUS file reset indicator:

Y = Reset the RCP STATUS file before the command executes.

N = Do not reset the RCP STATUS file.

n = Request system function. A value of 11 specifies that all terminals will be loaded.

Example 1

ADXCS20L Y 11

This command resets the RCP STATUS file before the system function is started. The requested system function is Load All Terminals.

On a system with the Multiple Controller Feature installed and running, terminals attached to each store controller can be loaded. For example, the terminals are attached to store controller CC and DD in a three-controller system. Controller EE is the store controller attached to the host line. You would send a COMMAND file containing the following:

CC::ADXCS20L Y 11

DD::ADXCS20L N 11

This COMMAND file would cause RCP to start the Load All Terminals program on the store controllers whose ID precedes the function name. The terminals attached to store controller CC and DD are loaded.

Loading terminals in a LAN environment is needed to support all terminals. ADXCS20L support only loads those terminals attached to the store controller where ADXCS20L is running.

Example 2

DD::ADXCS20L N 11

This COMMAND will cause RCP to start the Load All Terminals program on the store controller whose ID precedes the function name. All of the terminals that are reachable via the LAN by the DD controller will be reloaded. This example does not reset the RCP STATUS file.

Remote Set Terminal Characteristics Command

The function can also be started from the host. It is performed by the Remote Set Terminal Characteristics command to change a terminal configuration. You can send terminal configuration files from the host to the store controller as maintenance to be applied with ASM. Then, you can start the Remote Set Terminal Characteristics command from the host.

Each terminal to be reconfigured must be assigned a terminal number. The Remote Set Terminal Characteristics application runs when the command is accepted at the store controller. Any terminal

applications are stopped at this time. When Remote Set Terminal Characteristics begins to run, *Z100* is displayed on the terminals. If the command completes without errors, all terminals reload with the new configuration.

However, if there are serious errors (for example, an error writing to hard totals) the manual entry Set Terminal Characteristics is loaded on the terminals with errors and the operator must run the Set Terminal Characteristics command manually on the partner terminal. For other errors, the default applications are loaded and a message is logged in the system message file indicating the type of error encountered. You can fix the problem and send the request again from the host.

Note: If a partner terminal is also to be configured, it must be powered on at the time when the Remote Set Terminal Characteristics command is issued. If it is not powered on, the device group records are configured, but the default application is not configured.

The Remote Set Terminal Characteristics command enables a user to reconfigure terminals from the host.

Format
ADXCS20L *i* 12

Where:

i = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset STATUS file.

Example
ADXCS20L Y 12

This command resets the RCP STATUS file before the system function is executed. The requested system function is the Remote Set Terminal Characteristics.

Note: This command must be executed at the master store controller.

Set System Time and Date Command

This function enables an application to set the system-wide date and time. Changes made using this function are broadcast to all controllers and all terminals within the store system.

Format
ADXCS80L *i*

Where:

i = RCP STATUS file reset indicator, BACKGRND, or blank.
Y = Reset the RCP STATUS file and write output there.
N = Do not reset RCP STATUS file but write output there.
BACKGRND
Running in background (no output).
"blank"
Running in foreground (Command Mode) with output to the console.

command

One or more of the following separated by spaces:

- +1** Increment the current time by one hour.
- 1** Decrease the current time by one hour.
- B** Broadcast current time without altering.
- Q** No output (quiet mode).
- T HHMMSS**
Change the current time to HHMMSS.
- D YYMMDD**
Change the date to YYMMDD.

Note: This command will run only on the master controller if on a multiple controller network. Some commands are mutually exclusive (+1, -1, -B, -T) and only the last command is recognized. You must set the time if you set the date (that is, -D requires the -T). The order of the commands is not important. All of the commands implicitly broadcast their changes; you do not need to manually broadcast the time after making a change. You should use +1/-1 wherever possible to avoid inherent delays with setting the time directly. The system date and time can also be set using the system functions menu option. See the *4690 OS: User's Guide* for details on using this method.

Example

```
ADXCS80L Y +1
```

This command resets the RCP STATUS file and increments the current time by one hour. By default, the output of this application includes the time before execution and the time after execution if one of the parameters should cause the time to change.

File Compression/Decompression Command

Two files are associated with file compression and decompression:

- An alternate STATUS file that you can use instead of the RCP STATUS file
- A file containing the processing file name messages

You can specify that the alternate STATUS file, ADX_SDT1:ADXCSHAF.DAT, be used when the application using the ADXSTART function starts file compression. The alternate STATUS file allows the messages produced by file compression to be placed in the secondary STATUS file. For more information on starting RCP, see "RCP Commands" on page 136.

The processing messages are logged in the ADX_SDT1:ADXCS3PF.DAT STATUS file instead of in the RCP STATUS file. In a LAN environment, the STATUS file is a local file on the store controller that runs file compression.

The processing message file is reset when the STATUS file reset parameter is either Y or N. Any other value for the STATUS file reset parameter causes new messages to be appended to the existing RCP STATUS file. RCP pulls this file to the master as a report at the end of processing.

Format

```
ADXCS30L i n f s [d] [2]
```

Where:

- i =** STATUS file reset indicator:
 - Y =** Reset the RCP STATUS file before the command executes.
 - N =** Do not reset the RCP STATUS file.

- n* = Selects the options:
- 1 = File compression
 - 2 = File decompression
- f* = Type of files requested:
- 1 = Single file or wildcard (*) files
 - 2 = File named in the source is a list file (see “List File Format for File Compression/Decompression” on page 167 for details)
 - 3 = Source is a subdirectory
 - 4 = Source is a disk drive
- s* = Source file specification:
- For single files, you must specify the drive, subdirectory, file name, and optional file extension. The file name and optional file extension can have wildcards. Logical names are acceptable.
 - For a list file name, this specification must conform to the single file requirements. Wildcards are not acceptable when the list file name is given. See “List File Format for File Compression/Decompression” on page 167 for details.
 - For a subdirectory, you must specify the drive and subdirectory. Logical names are acceptable.
 - For a hard disk drive, the drive letter followed by a colon (:) is required.
 - When using option 4 on the *f* parameter (the source is a disk drive), the *s* parameter must contain a drive letter followed by a colon.

Note: You can specify a node name. If you do not specify a node, the local node ID is assumed.

- d* = Destination file specification:

Valid ways of identifying the destination files are:

- When the source is a file name, the destination can be one of the following:
 - Not specified
 - Drive
 - Drive and subdirectory
 - Drive, subdirectory, file name, and optional file extension
 - Subdirectory
 - Subdirectory, file name, and file extension
- When the source is a list file name, you cannot specify a destination. Any destination you specify is ignored. See “List File Format for File Compression/Decompression” on page 167 for details.
- When the source is a subdirectory, the destination must be one of the following:
 - Not specified
 - Drive
 - Drive and subdirectory
 - Subdirectory
- When the source is a drive, the destination must be one of the following:
 - Not specified
 - Drive
- When the destination is not specified, the destination is the same as the source.

Note: You can specify a node name. If you do not specify a node, the local node ID is assumed. Logical names are acceptable.

- 2 = The alternate STATUS file indicator. This parameter is optional. If you do not want to have this alternate STATUS file used, do not code this parameter.

Note: System limitations prevent using more than 46 characters for parameters on programs started using RCP. Therefore, if the parameters for a compression/decompression operation exceed 46 characters, use one of the following methods to shorten the command line:

- Use logical names for source and destination file names.

- Use a list file containing the parameters (which can exceed 46 characters) and possibly containing more than one request.

Example 1

```
ADXCS30L Y 1 1 ADX_IDT1:EAL*.DAT A:
```

This command compresses all files in the ADX_IDT1 subdirectory that start with EAL and that have an extension of DAT. These compressed files are placed in ADX_IDT1 subdirectory on the A: drive (a diskette drive). The RCP STATUS file is reset.

Note: ADX_IDT1: is a logical name for C:\ADX_IDT1\.

Example 2

```
ADXCS30L N 1 3 C:\ADX_IPGM C:\BACKIPGM
```

This command compresses all files in the ADX_IPGM subdirectory on the C: drive and places the compressed files into the BACKIPGM subdirectory on the C: drive. If the BACKIPGM subdirectory does not exist, it is created.

Example 3

```
ADXCS30L N 2 4 A: C:
```

This command decompresses all files on the A: drive (a diskette drive) and places the output on the C: drive. The output files are placed in the same subdirectories on the C: drive as the files came from on the A: drive. These subdirectories are created if they do not already exist.

List File Format for File Compression/Decompression

The list file can be built using any editor, including the DREDIX editor available on the operating system. Each entry in the list file must have the following format:

Format

```
f s [d] cr-lf
```

Where:

- f* = Indicates the type of files requested:
- 1** = Single file or wildcard files are requested.
 - 3** = The source is a subdirectory.
 - 4** = The source is a disk drive.

- If 2 is specified, an error results and processing continues with the next entry in the list file.
- s* = The source file specification:
- For single files, you must specify the drive, subdirectory, file name, and optional file extension. The file name and file extension can have wildcards. Logical names are acceptable.
 - For a subdirectory, you must specify the drive and subdirectory. Logical names are acceptable.
 - When using option 4 on the *f* parameter (the source is a disk drive), the *s* parameter must contain a drive letter followed by a colon.

Note: A node name can be specified. If no node is specified, the local node ID assumed.

d = The destination file specification.

Valid ways of identifying the destination file are:

- When the source is a file name, the destination must be one of the following:
 - Not specified
 - Drive
 - Drive and subdirectory
 - Drive, subdirectory, file name, and optional file extension
 - Subdirectory
 - Subdirectory, file name, and optional file extension
- When the source is a subdirectory, the destination must be one of the following:
 - Not specified
 - Drive
 - Drive and subdirectory
 - Subdirectory
- When the destination is not specified, the destination is the same as the source. Logical names are acceptable.

Note: You can specify a node name. If you do not specify a node, the local node ID is assumed.

cr-lf = Delimiters that indicate the end of an entry in the list file. These are the carriage return and line feed characters in ASCII. Most editors place these characters in the file when you press the *Enter* key.

The end of file (EOF) character is optional at the end of the list file.

Problem analysis data compression command

This command enables you to capture problem analysis data to a hard disk drive. The data is compressed in the zip format.

Create Problem Analysis diskette (CPAD) also uses list files to collect files. The OS provided list file, ADX_SDT1:ADXCSRXL.DAT contains a list of files to be collected. Users must not modify this file. It may be overwritten during future OS migrations and upgrades. Users may collect files by creating the file, ADX_IDT1:ADXCSRUL.DAT.

Format of ADXCSRUL.DAT file:

- Wildcards (*) and (?) in file entries are supported
- Logical names in file entries are supported
- The file entries are treated as optional
- All errors received when attempting to open the files in the list are ignored
- The files are collected whenever CPAD is run

Example entries for ADXCSRUL.DAT:

```
c:\program.log
ADX_SPGM:*.LOG
ADX_IDT1:ADXCSHCF.*
ADX_IPGM:???TS10L.SYM
ADX_UPGM:*.SYM
```

An ordered list of compressed output files is logged in the problem abstract file, ADX_SDT1:ADXSRCF.DAT. This list uses HCP file naming conventions. The compressed files in the list can be retrieved by a host site.

The problem abstract file also contains a summary of which reports and dumps were compressed, along with a summary of any requested problem analysis data that was not found.

Compressed output files are written to the ADX_SDT1 directory with the format ADXZxxxF.DAT. The xxx is a counter that starts with 000, and is incremented when an output file reaches its maximum size. If you plan to use HCP, it is recommended that each output file be one megabyte in size, reducing the amount of data to retrieve should a host transmission error occur.

Problem analysis report commands must be used to direct report output to disk before starting ADXCSR0L. System and performance trace files must be formatted before starting ADXCSR0L.

Format

ADXCSR0L *i n [d] [e] [x]*

Where:

i = STATUS file reset indicator:

- Y = Reset the RCP STATUS file before the command executes.
- N = Do not reset the RCP STATUS file.

n = Type of files selected:

1 - 63 = the sum $f_1 + f_2 + f_3 + f_4 + f_5 + f_6$,

where each f_n is selected from:

<i>Not applicable</i>	= 0
System Log Report	= 1
Terminal Dump	= 2
Controller Dump	= 4
System Trace Report	= 8
Performance Report	= 16
Module Level Report	= 32

d = Destination. This is an optional parameter:

- 4 = Hard disk drive C
- 5 = Hard disk drive D, or
- C: = Hard disk drive C
- D: = Hard disk drive D

e = Compressed file size. This is an optional parameter:

- 1 = 1024 KB
- 2-9 = Maximum number of megabytes per each output file
- * = All outputs to one file

x = Single keyword description for problem abstract. This is an optional parameter.

Example 1

ADXCSR0L Y 13 4 2 PMR56789

This example:

- Resets the RCP STATUS file
- Gathers the Controller Dump, System Log, and System Trace because $4 + 1 + 8$ equals 13
- Compresses output files to the C: drive
- Each output file is no larger than 2048 KB in size
- Problem abstract file contains PMR56789 keyword

It is assumed that the System Log and System Trace were previously formatted using ADXCSN0L and ADXCSM0L.

Example 2

ADXCSR0L N 16

This example:

- Does not reset the RCP STATUS file
- Gathers the Performance Report only
- Compresses output files to the default drive
- Each output file is no larger than the default size
- No keyword is placed in the problem abstract file

It is assumed that the Performance Report was previously formatted using ADXCSP0L.

Example 3

ADXCSR0L N 34 D: *

This example:

- Does not reset the RCP STATUS file
- Gathers the Module Level Report and Terminal Dump because $32 + 2$ equals 34
- Writes output files to the D: drive
- Compresses all output to one large file
- No keyword is placed in the problem abstract file

It is assumed that the Module Level Report was previously formatted using ADXCSS0L.

Extract Store-Specific Configuration Command

This command extracts store-specific configuration data from active files. The data can then be inserted into inactive files using the ADXSSC1L command. The extracted data is written to the

ADX_SPGM:ADXSSCDF.DAT file. After the initial execution of this command, you only need to run this command again to update the store-specific configuration data. However, you can run this command before each transfer of a configuration file.

Format

ADXSSC0L *i*

Where:

i = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.

Example

ADXSSC0L Y

This command resets the RCP STATUS file and extracts the store-specific data from the active files and writes the data to ADX_SPGM:ADXSSCDF.DAT.

Insert Store-Specific Configuration Command

This command writes the store-specific configuration data that was extracted using the ADXSSC0L command to the inactive files. This command enables you to transfer the same file to several stores without having to modify it first for store-unique data.

Format

ADXSSC1L *i*

Where:

i = STATUS file reset indicator:
Y = Reset the RCP STATUS file before the command executes.
N = Do not reset the RCP STATUS file.

Example

ADXSSC1L Y

This command resets the RCP STATUS file and writes the extracted store-specific data to the inactive files.

Controller BIOS Update Command:

- | This command reprograms the controller system's BIOS to the level shipped with 4690. The new BIOS level will be active after the next cold boot (note: the required reboot is not provided by this command).
- | Currently the command is supported on the following machine types/models: 4900-xx5.

Format

biosupdt.386

Starting RCP

You can start RCP from the host processor or from an application program. If you are using a LAN system, RCP must be run on the Master controller.

Starting RCP from the Host Processor

You can start RCP from the host processor by using one of the following methods:

- Issue the ADCS START USER PROGRAM command.
- Issue the NetView DM INITIATE FUNCTION command.
- Send a BIND request to the 4690. The program name ADXCSH0L is issued in the user data field of a BIND. Refer to your Systems Network Architecture (SNA) guide for information on a BIND.

You might need to consult your network coordinator for session or LU names to use with the ADCS START USER PROGRAM command. For information about specifying the session or LU name during configuration, see the *4690 OS: Planning, Installation, and Configuration Guide*. For information on starting SNA applications and the ADCS START USER PROGRAM command, refer to “Starting LU 0 Applications” on page 210.

When the RCP application (ADXCSH0L) is started, it reads the RCP SELECTION file, finds the appropriate RCP COMMAND file, and begins executing the commands in that file.

While RCP is running, no messages tell you when an RCP command is finished. However, you can learn the status of a command by looking at the STATUS file.

Starting RCP from an Application

You can also start RCP from an application. Consider the following when starting RCP from an application:

- The application can update the RCP SELECTION file, ADX_IDT1:ADXCSHCF.DAT, with the name of the RCP COMMAND file.
- The application can update or create the RCP COMMAND file in the ADX_IDT1 subdirectory.
- The application can issue the ADXSTART command to start RCP as a background job. No parameters need to be passed to RCP.
- RCP opens the RCP STATUS file with exclusive access. This means that no application should open the RCP STATUS file while RCP is running.
- The application can check whether RCP is running by opening the pipe, pi:ADXCSHEP. If the return code indicates *pipe not found*, the RCP is not running and the application can check the RCP STATUS file. If the return code indicates anything other than *pipe not found*, the application should delay for a short time and try again.

Starting Applications Using RCP

Running applications in the operating system requires knowledge of how the system starts the application and what items the application must consider to handle this mode of operation. This section discusses each way applications can be started and the implications to the application writer. This section applies only to applications running on a 4690 store controller. Terminal applications are not addressed.

Starting Applications on the 4690 Store System

There are three ways to start applications on the 4690 Store System: command line, background, and foreground.

Command line starting of an application occurs when you enter the name of the application to run, along with any parameters, in Command Mode. Starting applications from within BAT files has the same characteristics as starting applications from the command line.

Starting applications in the background can be done different ways. To start the applications in the background, you can:

- Configure the applications to be started from the background control screen by pressing **SysRq** and selecting option **b**.
- Start applications using ADXSTART from another application.
- Use RCP to start applications in the background.

Applications can also be started in the foreground. To start applications in the foreground, select option **2** (secondary application) on the store controller MAIN MENU panel. Then, choose an application to start. You must configure the applications listed on the secondary application screen before you can start them.

Each method requires the application writer to consider unique characteristics. These characteristics determine the way in which parameters are passed to the application and the type of I/O available to the application.

Passing Parameters to Main Routines

In the operating system, parameters are placed in the environment table. C language applications access this table through calls defined in the C language interface section of the *4690 OS: Programming Guide*. CBASIC language applications access this information by using the COMMAND\$ function (the command tail).

The command tail has two strings:

- One contains the name of the application started
- One contains the parameters

Note: This is different from the ARGV, ARGV format that is common in C language programming. Parameters coming into the main routine are always in ASCII format. This means any number coming into the application as a parameter to the main routine must be converted to a numeric data type by the application if it is needed for numeric operations.

Command Line Applications

Parameters are passed to the main routine of the application in the command tail when the application is started from the command line.

For example, you can start an application named APPLX from the command line with the following entry:

```
C:> APPLX PARM1 PARM2 PARM3 PARM4
```

The command tail string containing the name of the application would be "APPLX". The double quotes are not part of the string; they show the extent of the string. The command tail string containing the parameters would be:

```
"PARM1 PARM2 PARM3 PARM4"
```

Note: The number of spaces between parameters remains exactly as they were entered on the command line. However, if the user entered several blank characters after the last parameter, these spaces would not appear in the parameter list string.

Input can come from Display Manager screens or from the basic I/O functions of the programming language in addition to the command tail. Output can go anywhere that the application wants to direct it. Both the keyboard and the video display are available to the application.

Background Applications

RCP Applications

As with 4680 or 4690 applications, parameters are passed in the command tail when the application is started in the background by RCP. For example, you can start an application named APPLY from RCP using the COMMAND file containing the following line.

```
APPLY PARM1 PARM2 PARM3 PARM4
```

The command tail string containing the name of the application would be "APPLY". The double quotes are not part of the string; they show the extent of the string. The command tail string containing the parameters would be:

```
"BACKGRND PARM1 PARM2 PARM3 PARM4 "
```

The operating system inserts the BACKGRND parameter into the command tail. This parameter is inserted at the beginning of the command tail parameter string to inform the application that it was started in the background. This is important because background applications do not have keyboards available for input and they do not have video screens available for output. However, the background application does have a status line on the background status screen that you can use to write messages.

Also, note that the number of spaces between parameters is reduced to one. RCP does this to ensure the maximum number of characters are passed to the application because there is a 45-character limit to the number of characters passed to the application when it is started in the background. In the example above, part of PARM4 would not have been passed to the application if the space reduction had not taken place.

Also, note that there is a space after the last parameter. If you entered several blank characters after the last parameter, only one space would appear at the end of the parameter list string.

Return codes from applications are provided as either four-byte integers or hexadecimal error codes that are significant in RCP because RCP will put the return code each application ended with in the RCP STATUS file. RCP notes the return code from each application RCP starts and, when RCP ends, it will end with the most negative return code from all of the applications RCP started in one session. RCP does this because NetView Distribution Manager operating at a host machine receives a return code from the 4690 store controller where RCP is operating. This action allows a host application to react to failing return codes from an RCP session.

If an application ends successfully, the return code from that application must be zero for it to be recognized as a successful execution. When an application ends before completing the function it was supposed to accomplish, the application should end with a negative return code.

Background Control Screen Started Applications

Applications started from the background control screen are similar to applications started by RCP. The only difference is that spaces between parameters are placed in the command tail parameter string just as they were entered during configuration of the background control screen applications. As with other 4680 or 4690 applications, parameters are passed in the command tail when the application is started from the background control screen.

For example, you can start an application named APPLY from the background control screen entry containing the following line:

```
APPLY PARM1 PARM2 PARM3 PARM4
```

The command tail string containing the name of the application would be "APPLY". The double quotes are not part of the string; they show the extent of the string. The command tail string containing the parameters would be:

```
"BACKGRND PARM1 PARM2 PARM3"
```

The operating system inserts the BACKGRND parameter into the command tail. This parameter is inserted at the beginning of the command tail parameter string to inform the application that it was started in the background. This is important because background applications do not have keyboards available for input and they do not have video screens available for output.

Note that the spaces between parameters are maintained. In this example, the PARM4 parameter would not be passed to the application because it is beyond the 45-character limit for background application parameter lists.

Background applications cannot have screens or keyboards associated with them. This reduces the possibilities for input and output. Input can only come from places not requiring an interactive user (for example, disk files). Output cannot be placed on any screen.

RCP STATUS File: The RCP STATUS file name is ADXCSHSF.DAT. The file resides in the ADX_SDT1 subdirectory and contains processing messages and error messages from each application that RCP started in a run. The STATUS file is the main form of communication between you and the background application. Each application that can be started in the background puts four types of messages into the RCP STATUS file.

The first message type identifies the application that is running and the parameters passed into the application. The format of this entry is:

- *tt:tt:tt* (the time)
- RUNNING
- application
- parameters passed to application

The time field contains hours, minutes, and seconds. This entry is consistent with the same entry placed in the RCP STATUS file by the 4680/90 system applications.

The second message type is a processing message. Some applications do a variety of tasks and it might be important for you to see what the application accomplished. For example, this type of message can be a file name when a list of files in a subdirectory are being processed by the application. This type of message is a free form message.

The third type of message is an error message. Any time the application recognizes errors, a message can be written to the RCP STATUS file. This type of message is a free form message.

The fourth type of message written to the RCP STATUS file is a completion message. It's purpose is to tell you how the application completed: successfully, failed, or only did part of the requested function. The format of the completion message is:

- *tt:tt:tt* (the time)
- SUCCESS, FAILURE, or PARTIAL
- application
- parameters passed to application

The time field contains hours, minutes, and seconds. This entry is consistent with the same entry placed in the RCP STATUS file by the 4680 or 4690 system applications.

Normally, the application writing to the RCP STATUS file will write its messages at the end of the existing RCP STATUS file. This allows any application that was previously run in this RCP session to keep its messages in the RCP STATUS file for someone to review. If no RCP STATUS file exists, the application should create one in the ADX_SDT1 subdirectory.

If the system is a LAN system, the application should use the RCP STATUS file on the master store controller, regardless of the node on which the application is actually run.

Foreground Applications

Applications started in the foreground are different from applications started using the other methods. These are applications that are started by choosing an option on a 4690 screen. You configure these applications on the secondary application screen.

No parameters can be configured for applications started in the foreground. However, that does not mean there are no parameters for the application to consider. For example, if you start an application named APPLZ from the secondary application screen, the command tail string containing the name of the application would be "APPLZ". The double quotes are not part of the string; they show the extent of the string. The command tail string containing the parameters would be:

```
"FOREGRND"
```

The FOREGRND parameter informs the application that it was started as a foreground application and there is a video screen and keyboard associated with it. This is important because foreground applications typically are interactive applications and communicate with you using the video screen and keyboard.

Foreground applications have the most I/O flexibility. Input can come from any place and output can go anywhere.

Application Programmer Considerations

If you intend for an application to be run only in one environment (command line, foreground, or background) then only characteristics of that environment need to be addressed. However, if the application is intended to be run in more than one environment, the application must be flexible enough to handle the characteristics of each intended environment. Many of the 4680 or 4690 system applications are written in such a manner. For example, if you intend to run the application from each of the three environments, the following characteristics apply:

- Inserted Parameters

The application should be able to determine how it was started by looking at the first parameter. If the parameter is FOREGRND, it was started from the secondary application screen. If the parameter is BACKGRND, it was started using one of the methods starting background applications. If the first parameter is neither FOREGRND nor BACKGRND, or there are no parameters, the application was started from the command line.

This assumes the application did not specify the first parameter for you to use to be FOREGRND or BACKGRND.

- Parameter List Handling

The application extracting the parameters as they are encountered parses the parameter list string. If the parameter list string is used as a fixed structure, parameters will not be correctly used by the application when the application is started in each of the three possible environments. This is due to the differing ways of handling spaces between parameters and the differing number of parameters. This method of obtaining the parameters eliminates any dependency that the parameters be presented in a certain format and allows the application to get the same parameters in the same order regardless of the environment.

IPL Processing

When processing is needed during an IPL of the 4690 store controller, RCP can provide support using a special SELECTION file named ADX_SDT1:ADXCSHDF.DAT. When RCP is started during the IPL, it checks for the SELECTION file first. If the special SELECTION file exists, RCP runs all commands listed in the COMMAND files. If any command in the SELECTION file causes the store controller to IPL, when RCP is started again it runs the next command until all commands have been executed.

To activate the unique IPL processing, you must configure RCP (ADXCSH0L) to be started in the background on IPL of the store controller.

Note: You must code the IPL parameter before RCP knows that this is the special IPL type of processing.

For a LAN, the master store controller is the only store controller that RCP is started on for the IPL processing. RCP is not started on every node in a LAN environment during the IPL of the store controller.

Processing Multiple Commands and COMMAND files: RCP enables you to request more than one command in a COMMAND file and more than one COMMAND file name in a SELECTION file. Without the RCP IPL parameter, the IPL of a store controller causes any commands that followed the IPL command to be ignored. With RCP IPL processing, however, RCP issues all commands that are contained in each COMMAND file listed in the SELECTION file, even if the commands cause a store controller to IPL. This situation is possible because RCP starts on every IPL of the 4690 store controller and continually tracks the commands that are issued. Tracking enables RCP to issue commands that were not issued when RCP started.

When RCP started because of an IPL, you are notified in the RCP STATUS file, ADX_SDT1:ADXCSHSF.DAT. When more than one command causes an IPL, RCP continues to execute the next command on the next IPL until all commands are executed.

The following example illustrates the interaction between multiple commands and IPL processing:

- ADXCSHCF.DAT contains: *USERCMD.DAT*.
- USERCMD.DAT contains:
 ADX_UPGM:USERPROG
 ADXCST0L N 3 M
 ADXCSS0L N 1 M

In this example, the goal is to execute a user-written program, USERPROG, followed by the Apply Software Maintenance command, ADXCST0L. After maintenance has been applied, the ADXCSS0L command generates a module level report.

The USERPROG generates ADXCSHDF.DAT before ending and contains PREIPL.DAT. The PREIPL.DAT file contains the command:

```
MOVEDATA ADX_UDT1 ADX_UBUL
```

When the ADXCST0L program runs, the store controller IPLs. RCP is started after the maintenance has been applied. The MOVEDATA program is executed first, followed by the execution of ADXCSS0L. RCP finds the ADXCSHDF.DAT file created by the USERPROG program.

If RCP was not running immediately before the 4690 store controller IPLed, RCP does not issue the list of commands identified by the ADXCSHCF.DAT file. RCP issues these commands only when RCP is running immediately before a store controller IPL.

Retrieving Reports Formatted by RCP

You can retrieve a formatted report by using your transmission facility at your host processor and specifying the appropriate name of the report you are retrieving. You can use the HCP or the Remote Change Management Server (RCMS) to send the formatted report results to the host processor. Table 43 shows which store controller files contain the formatted report results.

Table 43. Reports Formatted by RCP

Report	File Name	Subdirectory
System Log	ADXCNSRF.DAT	ADX_SDT1
Trace Report	ADXCSTMF.DAT	ADX_SDT1
Configuration	ADXCSDTF.DAT	ADX_SDT1
Module Level Report	ADXCSSDF.DAT	ADX_SDT1

Table 43. Reports Formatted by RCP (continued)

Report	File Name	Subdirectory
Dump Summary	ADXCSLRF.DAT	ADX_SDT1
Tape STATUS file	ADXCSVDF.DAT	ADX_SDT1
Audible Alarm Status	ADXCS1RF.DAT	ADX_SDT1
Optical STATUS file	ADXCS6DF.DAT	ADX_SDT1

File Names for Report Files

Table 44 shows the 6-character HCP file names for all report files used with RCP (except the RCP COMMAND file) if your host processor communicates with the store controller through the HCP. You can choose your own HCP file name for the RCP COMMAND file. Refer to “HCP File Naming Conventions” on page 109 for information needed to choose a valid HCP name.

Table 44. HCP File Names for Report Files Used with RCP

File Name	HCP Name
System Log	CSNRF&
Trace Report	CSMTF&
Configuration	CSDTF&
Module Level Report	CSSDF&
Dump Summary	CSLRF&
Tape STATUS file	CSVDF&
Audible Alarm Status	CS1RF&
RCP STATUS file	CSHSF&
RCP SELECTION file	CSHCF% (PC)* if file data is 4690-compatible. CSHCF\$ (PRINT)* if file data is EBCDIC and requires translation.
Optical STATUS file	CS6DF&

Applying Software Maintenance from a Host Site

Software maintenance can be activated from a remote site and from the local site.

See the *4690 OS: User's Guide* for more information on applying software maintenance locally.

To use ASM from a remote site, use the following procedure.

At the Host Store Controller for the 4690 Store System:

1. Use ASM to transfer all Toshiba maintenance from the maintenance package into the ADX_nMNT subdirectory.
 - If you are building your own load module, go to step 2.
 - If you are not building your own load module, go to step 9.
2. Copy the necessary application source and object modules into the ADX_UPGM subdirectory.
3. Code the user modifications.
4. Compile and link the application with user modifications.
5. Run a Report Module Level. This report helps determine the maintenance level of the modules in the ADX_nMNT subdirectory for the product to which you are applying user modifications. If no maintenance is pending, use the information found in the current directory.

6. Use the Build Software Maintenance Control File utility to build a user Product Control File. See the *4690 OS: User's Guide* for more information about this utility.
7. Copy the new .286 files from the ADX_UPGM subdirectory onto the diskette containing the user Product Control File.
8. Using ASM, transfer the user maintenance from the diskette.

At the Host Processor:

9. Retrieve each file from the ADX_nMNT subdirectory and load onto the host processor.

At the Host Store Controller for the 4690 Store System:

10. Activate the maintenance using ASM. Your store controller automatically IPLs.
11. Test the maintenance to verify it works properly.
12. When the maintenance is tested and verified, you can leave it in test mode or you can accept the maintenance.
13. If you have not created an RCP COMMAND file, do so at this time as described in "The RCP COMMAND File" on page 132 and "Apply Software Maintenance Command" on page 154. The parameters you use should indicate you want to accept maintenance for the product.
14. Retrieve the RCP COMMAND file created in step 13 to the host processor.

At the Host Processor:

15. Establish a communications session with the remote store controller. Transmit the RCP COMMAND file and each file you retrieved from the ADX_nMNT subdirectories to the remote store locations. This step places the files in the proper ADX_nMNT subdirectory on the remote store controller.
16. Transmit the RCP COMMAND file to the remote store locations.
17. To activate the maintenance, run ASM from the host processor by starting RCP as described in "Starting RCP" on page 172. The remote store controller automatically IPLs.
18. When communications with the store controller is reestablished, retrieve the RCP STATUS file from the store controller to determine the success of the maintenance you applied. The name of the RCP STATUS file is ADX_SDT1:ADXCSHSF.DAT. For a more complete verification of the success, run a Report Module Level.
19. Repeat this procedure as required if all maintenance subdirectories were not transmitted at the same time.

Chapter 8. Using Communications and Systems Management

The operating system supports host Communications and Systems Management (C & SM) facilities for alert messages. The system can support multiple SNA links. The C & SM support can be optionally configured to function on any subarea link. However, only one link at a time can be a C & SM link.

The alert interface in the host is NetView. The operating system sends alert messages for errors detected by the system and for exception conditions in the store controller and in the terminal. In addition, your applications in the store controller and in the terminal can request the operating system to send an alert message. The operating system sends only the alerts that occur when the host line is active.

NetView provides an operator interface for looking at the alert messages and obtaining additional information about the causes of alerts.

Note: See the ALERTS keyword in the *4690 OS: Planning, Installation, and Configuration Guide* for information about selecting the Alert Resource field format to be used.

System Alerts

The operating system sends alerts for a subset of the system log entries. The alerts are defined to assist with the identification of hardware and software problems. System alerts are sent when system events are logged and when application programming support is required. The setting of the severity code for which messages are displayed on the system message display does not affect which alerts are sent to the host. See the *4690 OS: Messages Guide* for additional information.

User-Application Alerts

The terminal and store controller applications can record events in the application event log and send an alert to the host by using the ADXERROR function. An event number is used to uniquely identify the event. The event number can be 1 through 127. Because some of the log entries do not require an alert, only part of the event number range causes alerts. For store controller applications, a log entry is made and an alert is sent for the event numbers of 64 through 79.

Table 45 lists the alert numbers that correspond with master controller event numbers.

Table 45. Alert Numbers for Master Controller Event Numbers

Master Controller Event Number (Decimal)	Alert Number (Hexadecimal)
64	74
65	75
66	76
67	77
68	78
69	79
70	7A
71	7B
72	7C
73	7D
74	7E
75	7F
76	80

Table 45. Alert Numbers for Master Controller Event Numbers (continued)

Master Controller Event Number (Decimal)	Alert Number (Hexadecimal)
77	81
78	82
79	83

Table 46 lists the alert numbers that correspond with subordinate controller event numbers.

Table 46. Alert Numbers for Subordinate Controller Event Numbers

Subordinate Controller Event Number (Decimal)	Alert Number (Hexadecimal)
64	84
65	85
66	86
67	87
68	88
69	89
70	8A
71	8B
72	8C
73	8D
74	8E
75	8F
76	90
77	91
78	92
79	93

For terminal applications, a log entry is made and an alert is sent for event numbers of 64 through 81. Table 47 lists the alert numbers that correspond with terminal event numbers.

Table 47. Alert Numbers for Terminal Event Numbers

Terminal Event Number (Decimal)	Alert Number (Hexadecimal)
64	94
65	95
66	96
67	97
68	98
69	99
70	9A
71	9B
72	9C
73	9D
74	9E

Table 47. Alert Numbers for Terminal Event Numbers (continued)

Terminal Event Number (Decimal)	Alert Number (Hexadecimal)
75	9F
76	A0
77	A1
78	A2
79	A3
80	A4
81	A5

The operating system prepares the alert from the system information (such as date and time) and from the values provided in the application event log message. Your application can provide up to 10 bytes of data as a value that is included in the log. If the event number causes an alert, the 10 bytes must be ASCII characters because they are used as product-unique text data in the text message subvector of the alert.

Setting the severity code to determine which messages are displayed on the system message display has no effect on which application alerts are sent to the host. See the *4690 OS: Programming Guide* for an example of how to log an application event using ADXERROR.

Vital Product Data File

When alerts are sent to NetView in the network management vector transport (NMVT) request/response unit (RU), they contain information that identifies the product by its machine type, model number, and serial number. In the case of the Mod1 terminal, this data is stored in the totals retention RAM.

Special routines in the Diagnostic Monitor read this data from the RAM and store it in the Vital Product Data (VPD) file, ADXCSCVF.DAT, at the store controller.

The VPD file is a keyed file created by the configuration process. The record keys for this file are the assigned terminal numbers or controller IDs. The file contains 100-byte records for each terminal and store controller identified during configuration.

Records in the VPD file have the following format:

- Bytes 0 - 4** Record key
- For a store controller record:
 - bytes 0 - 1** Controller ID (ASCII, value range CC-ZZ)
 - bytes 2 - 4** ASCII, value 000
 - For a terminal record:
 - bytes 0 - 1** ASCII, value 00
 - bytes 2 - 4** Terminal No. (ASCII, value range 001-999)
- Bytes 5 - 11** Machine Type/Mod. No. (ASCII *nnnnaaa*)
- nnnn* = 4 numeric digits
 - aaa* = 3 alphanumeric characters
- Bytes 12 - 19** Serial No. (ASCII *mm-sssss*)
- For a store controller record, *nn-nnnnn* (ASCII, seven-digit Personal System/2 (PS/2) serial no.)
 - For a terminal record, *mm-sssss* (ASCII)
 - bytes 12 - 13** *mm* — Plant of Manufacture
 - byte 14** Dash character (-)
 - bytes 15 - 19** *sssss* — Sequence No.

Bytes 20 - 26 Planar Card EC No. (ASCII)
BIOS level of machine (supports all)

Bytes 27 - 33 Power Supply EC No. (ASCII)
This does not exist on a 4694 machine and appears as dashes (--)

Bytes 34 - 36 4683 Model 001 80286 ROS ID/EC No. pair (binary)
This appears as zeros on a 4694 machine

Bytes 37 - 96 30 8051 ID/EC Nos. (binary)

Bytes 97 - 99 Reserved

MAC Address Data File

The operating system provides a data file for functions like Wake on LAN that require a mapping between 4690 OS terminal numbers and MAC addresses.

The file, ADX_SPGM:ADXCVMAC.DAT is a keyed file with a 128-byte record length and a 3-byte key length. The key is the 3-digit ASCII representation of a terminal number. For example, the key for terminal number 1 is 001 as three ASCII bytes.

The MAC file is created during controller IPL and is populated by individual terminals as they come online.

Records in the MAC file have the following format:

Bytes 0 - 2 Record key (ASCII, value range 001-999)

Byte 3 WoL Supported (byte, value range 0-1)

Bytes 4 - 9 MAC address (Byte Array[6])

Bytes 10 - 127
Reserved

The MAC address recorded in this file is the "burned-in" MAC address of the POS terminal Ethernet port.

The byte labelled "WoL Supported" reports 0 if Wake on LAN is not supported by this terminal and reports 1 if Wake on LAN is supported by this terminal.

At present, only Enhanced Mode terminals are recorded in this file. Additional entries or record data may be added in the future.

The operating system Keyed File Utility can convert the keyed file to a flat file, if an alternate format is required. If so, remember to refresh the flat file periodically because the data changes as POS terminals are added or renumbered and hardware replacement procedures alter the effective MAC address of a given terminal number.

Regarding reverse lookup, entries are added or refreshed periodically as terminals reboot; entries are never removed. As a result, program processing must take into account that obsolete records might exist if terminal numbers are abandoned over time and those obsolete records might result in the appearance of multiple terminal numbers associated to the same MAC address.

The file is oversized to accommodate the full range of possible terminal numbers 000-999 with additional space to minimize the possibility of overflow records and to allow additional records to be added in the future without requiring the file to be resized.

Part 2. 4690 Communication Programming

Chapter 9. Designing an Application for 3270 Emulation

For information about how to use 3270 emulation at the terminal, see the *4690 OS: User's Guide*.

Using the Application Programming Interface

The application programming interface (API) provided by the 3270 emulator enables an application to execute all of the functions that an operator would normally perform. A typical session for an operator can involve:

- Starting the 3270 emulator (store controller only)
- Logging on to a host application and then repeating the following steps until logging off:
 - Receiving a panel of data from the host application (select options, enter values, and so on)
 - Sending the responses to the host by pressing **Enter** or a program function key (PF key)
- Logging off from the host application
- Terminating the 3270 emulator (store controller only)

You can use the 3270 API to automate all or part of this process with the following restriction on the 4690 store controller:

If the session is to be shared between 3270 API applications and users, the emulator must be started in the foreground from the command line and must be ended in the foreground by pressing **Exit**.

For example, on the 4690 store controller, you can provide two 3270 API applications, one to log on and the other to log off. You start the 3270 emulator, and then start the logon application. This application waits for the host's welcome panel at the start of a session, and keys in the logon commands for the operator. Once logged on, the application turns over control to the operator. When the operator has completed using the session, the operator starts the logoff application that completes the session and logs off the user from the host application. The 3270 emulator continues to execute in the operating system until the operator stops it by pressing **Exit**.

While the application is using the 3270 emulator, you can only stop the emulator by pressing **Exit**. Any other keystrokes entered at the keyboard are ignored, and the panel is not updated. When the operator has control of the 3270 emulator, it cannot be accessed by any 3270 application using the API.

On the terminal, user exits in the sales applications can access the 3270 API. For example, a user exit invoked when a terminal operator signs on to the sales application can be used to log the user on to the 3270 session at the same time. In this example, the 3270 application using the API is terminal sales.

In another example on the 4690 store controller, the 3270 API application can be written so that operator intervention is not needed. A 3270 API application can:

- Start the emulator
- Detect the welcome panel and log on to the host
- Search each panel of data received from the host to determine the responses required
- Type responses
- Log off and stop the 3270 emulator

In the previous example, the 3270 emulator executes in the background and does not need access to the panel or keyboard.

Session Identifiers

Both the store controller and the terminal implement the same API. There is only one copy of the 3270 emulator active that the application can access. On the store controller, a mechanism is needed to relate the 3270 API application to the particular copy of the 3270 emulator that it is accessing. This relation is established using the following guidelines:

- The application and the emulator must run on the same store controller.
- You must define a session identifier when the 3270 emulator is started. The session identifier is a single letter from A to Z or a to z.
- Two 3270 emulators on the same store controller cannot use the same session identifier. The second emulation to start with a session ID already in use logs a W845 error. The second emulation initializes and accepts operator use, but the emulation is not accessible to API applications.
- Each 3270 API application uses the session identifier to specify which 3270 emulator it is going to access.
- If you start a 3270 emulation on the 4690 store controller without specifying a session identifier, the default used is null. No applications can access the emulator.

On the point-of-sale terminal, no session identifier is needed.

3270 API

This chapter refers to the API subprograms as verbs. A *verb* is defined as a function call. The API application issues a *verb*. For example, the API includes the S3.CONNECT subprogram. To issue the Connect *verb*, the application sets up the parameters for the subprogram and calls S3.CONNECT.

This chapter also refers to the 3270 panel image as the *presentation space* (PS). However, while the emulator is controlled from the API, the 3270 panel image is never actually displayed on a physical console device such as the store controller console or the terminal monitor. A message is displayed on the panel indicating that the session is under application control. Offsets within the PS are given as absolute values between zero (0) and 1919. Zero is located at the top left corner of the panel, and 1919 is located at the bottom right corner.

Starting 3270 Emulation

An application executing in a store controller can use the API in the following ways:

- If a 3270 session is to be serially reused by an operator and an application or applications, the operator must start the 3270 emulator process before the application's first attempt to access the API.
- If a 3270 session is used exclusively by an application in only an API session, the 3270 emulator process is started by the Connect *verb* (S3.CONNECT) in the API.

The reason for this distinction is that if an application is going to share a 3270 session with an operator or other applications, the 3270 session must be accessible to all. To be accessible to an operator, the 3270 emulation process must run in the foreground with control of the keyboard and panel.

If a 3270 session on the store controller needs access to the operator panel and keyboard I/O, the emulator must be started from the foreground. Conversely, if access to the 3270 session is only through the API session, the application can start the 3270 session in the background.

Starting 3270 Emulation from the Command Line (Store Controller Only)

When starting the 3270 emulation process, the operator specifies a session ID in the command line parameters. The session ID is represented by a single letter. The session ID is used to create three unique pipe names. The pipes allow communication between the 3270 emulation process and an application process. The session ID is the fourth component of the command tail.

For example:

```
>adxhs301 adxlxddn, 3270link,prn1:,X
```

The session ID is case-sensitive. The session ID can be defined as a single letter from A to Z or from a to z.

Note: The application that connects to this example of 3270 emulation must be written specifically for this ID. The session ID can be omitted if the session is never going to be accessed by an application.

When the emulation is active, the operator can press Tempexit to enable the application to use the session. If the application attempts to use the session before the operator presses Tempexit, then the connect function sends a negative return code. The application should be written to retry the connection until successful.

Starting 3270 from the API (store controller only)

The application starts the emulator and specifies the session ID in the parameters of the S3.CONNECT verb. The verb starts a copy of the emulator using the session ID and other parameters to build a command tail for the emulator. Starting the emulator this way is the same as for an operator-initiated program except the session automatically starts as an API session. Therefore, you need not use the Tempexit key.

Using a 4680 BASIC interface to access 3270 emulation

The 3270 Emulation feature API is accessible from 4680 BASIC programs. This section describes the functions made available in the API, and the 4680 BASIC definitions.

All of the API verbs are synchronous. The application is suspended until the requested function is complete.

API verb timeouts

All API verbs execute by sending a pipe message to the 3270 emulator and waiting for a pipe message response. The timeout on the wait for each verb depends on the verb type as indicated in Table 48.

Table 48. API verb timeouts

Verb type	Timeout
CONNECT	16 minutes
WAIT type 0	4 minutes
WAIT type 1	4 minutes or specified time (whichever is larger)
WAIT type 2	40 minutes
All other verbs	4 minutes

If a response is not received in the specified timeout, a -1 return code is passed to the application program. After a timeout occurs, the application using the API is disconnected from the 3270 emulator.

API verbs

This section contains descriptions of the API verbs that are used with 3270 emulation.

All of the API verbs are synchronous.

Connect

The Connect verb (S3.CONNECT) requests connection to a specified session.

Connect contains some optional parameters. However, the API must specify values for each parameter for the application to compile. The emulator ignores the parameters that are not needed.

On the terminal: None of the parameters are needed. The parameters have been retained for terminal applications to remain consistent with the store controller.

None of the parameters are needed on the terminal because the emulator is always started either at IPL by the operating system, or when the 3270 application ADXZE30L.286 is loaded. The emulator receives the information (such as the link name, whether it is to be shared, and the node name) from the startup parameters and does not use the values passed in the Connect verb.

On the store controller: If the 3270 emulator is already active, meaning it is not being started by this application, only the following parameters are needed:

SESSION.ID\$

SHARED% (must be 1)

If the 3270 emulator is being started by this application, then all of the parameters are needed, and SHARED must be set to 0 even if the 3270 emulation is shared with other applications.

Syntax for connect: An example of the Connect verb follows:

```
SUB S3.CONNECT(SESSION.ID$,      \
                      SHARED%,   \
                      NODE.NAME$, \
                      LINK.NAME$, \
                      PRINT.NAME$, \
                      MESSAGE.TEXT$, \
                      RETURN.CODE%) EXTERNAL \
INTEGER*2 RETURN.CODE%
STRING    SESSION.ID$
INTEGER*2 SHARED%
STRING    NODE.NAME$
STRING    LINK.NAME$
STRING    PRINT.NAME$
STRING    MESSAGE.TEXT$
END SUB
```

Input parameters for connect:

SESSION.ID\$

Significant on store controller only.

Identifier of session to which connection is requested. The session ID is case sensitive. If the application is to use an existing 3270 emulation process, this parameter must match that specified when the operator or another application started the 3270 emulation.

The session identifier is a single letter from A to Z or from a to z.

SHARED%

Significant on store controller only.

0 Session is not to be shared with the operator, and is being started by this API application.

- 1 Session is to be started with operator or has been started by another application.

NODE.NAME\$

Significant on store controller, non-shared sessions only.

Name of node on which the SNA driver is running. Equivalent to command line parameter for 3270 sessions started by operator.

LINK.NAME\$

Significant on store controller, non-shared sessions only.

Name of link to be used by 3270 session. Equivalent to command line parameter for 3270 sessions started by operator.

PRINT.NAME\$

Significant on store controller, non-shared sessions only.

Name of printer to be used as local copy device by 3270 session. Equivalent to command line parameter for 3270 sessions started by operator.

MESSAGE.TEXT\$

Significant on store controller, non-shared sessions only.

Text to be displayed on BACKGROUND APPLICATION panel for 3270 emulator entry.

Return codes for connect:

RETURN.CODE%

- 0 Connection successful.
- 1 Function did not complete successfully due to one of the following system errors:
 - Unable to open pipes.
 - No response from 3270 emulation.
 - Error reading or writing pipe.
 - 3270 emulation process failed to start.
- 3 Incorrect parameter specified.
- 4 Session in use by another application.
- 5 Request rejected by 3270 emulation.

Output parameters for connect: There are no output parameters associated with the CONNECT verb.

Disconnect

The Disconnect verb (S3.DISCONNECT) requests disconnection from the session.

The terminate option is provided so that the application can choose whether to end the 3270 emulator that deallocates resources on the store controller, or to leave the emulator running. The terminate option has no significance for the terminal.

If the emulator is left running, another application or the same application can, at a later time, successfully issue an S3.CONNECT verb. If the emulator is shared, the operator can access the session by hot-keying into its window and pressing Online.

If the emulation was started by the operator from the command line (for the controller only) as a foreground application, then it cannot be terminated by the application using the API. The operator must end the emulation by pressing Exit.

Syntax for disconnect: An example of the Disconnect verb follows:

```

SUB S3.DISCONNECT(TERMINATE%, RETURN CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
INTEGER*1 TERMINATE%
END SUB

```

Input parameters for disconnect:

TERMINATE%

Flag: Significant on store controller, non-shared sessions only.

- | | |
|----------|---------------------------|
| 0 | Do not terminate session. |
| 1 | Terminate session. |

Return codes for disconnect:

RETURN.CODE%

- | | |
|-----------|--|
| 0 | Disconnection successful. |
| -1 | Function did not complete successfully due to one of the following system errors: <ul style="list-style-type: none"> • No response from 3270 emulation • Error reading or writing pipe |
| -2 | API not connected. |
| -3 | Incorrect parameter specified. |

Output parameters for disconnect: There are no output parameters associated with the Disconnect verb.

Wait

The Wait verb (S3.WAIT) checks the session to see if it is ready to receive keystrokes by simulating an operator waiting while the keyboard is locked.

In 3270 emulation, when Enter is pressed and data has been sent to the host, the keyboard is locked until the host responds and unlocks it. There are other conditions in the 3270 emulation system where the keyboard is locked. The timed wait returns before the specified timeout value if the host is ready to receive keystrokes. The operator can enter data only while the keyboard is unlocked.

The Wait function returns when the keyboard is unlocked. At this time, the next panel is not required to be displayed, depending on the host application. It is not unusual for the host to lock and unlock the keyboard several times between updates to the panel. A Wait is required for each lock and unlock cycle that the host completes.

For example, many installations initially display a selection menu when logging onto a host computer that lists the applications from which the user can choose. When the user selects a choice from this menu, the LU is unbound and then rebound to the chosen application. During this time, the keyboard is locked, unlocked, locked again, and finally unlocked to wait for user input. Waiting for the keyboard to be unlocked is not always sufficient to tell the application when data can be entered. Instead, the application can periodically check the screen buffer using the Locate-String verb for a string that appears when the host application has completed sending each screen. Note that the screen might be sent in segments and the last segment containing the string being checked for might not be at the bottom of the screen. The programmer must know the format of the screen so that the application can check the correct location. See "Locate-string" on page 193 for a description of the Locate-String verb.

The timeout is provided to avoid deadlocks in case of severe errors in the system preventing the 3270 emulator from ever resuming.

Note: If the application issues one of the verbs that simulates pressing keys (for example, Send-Key and Send-String), while the keyboard is locked, an error message is displayed on the operator guidance line. The operator guidance line must be cleared by pressing Reset. Applications using the 3270 API must issue Wait verbs frequently.

Syntax for wait: An example of the Wait verb follows:

```
SUB S3.WAIT(WAIT.TYPE%, WAIT.TIME%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
INTEGER*1 WAIT.TYPE%
INTEGER*2 WAIT.TIME%
END SUB
```

Input Parameters for Wait:

WAIT.TYPE%

Flag:

- | | |
|----------|-----------------------|
| 0 | Immediate return |
| 1 | Timed wait |
| 2 | Wait until successful |

WAIT.TIME%

Timeout in seconds. Valid only when WAIT.TYPE% is specified as **1**.

Return codes for wait:

RETURN.CODE%

- | | |
|-----------|---|
| 0 | Successful function. |
| -1 | Function did not complete successfully due to one of the following system errors: <ul style="list-style-type: none">• No response from 3270 emulation• Error reading or writing pipe |
| -2 | API not connected. |
| -3 | Incorrect parameter specified. |
| -4 | Request rejected by 3270 emulation. |
| -5 | Timer expired. |

Output parameters for wait: There are no output parameters associated with the Wait verb.

Locate-string

The Locate-String verb (S3.LOCSTR) searches for the first occurrence of a string starting from the current cursor position in the PS and moves the cursor to the start position of the string, if a string is found.

Syntax for locate-string: An example of the Locate-String verb follows:

```
SUB S3.LOCSTR(SEARCH$, DIRECTION%, OFFSET%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
STRING SEARCH$
INTEGER*1 DIRECTION%
INTEGER*2 OFFSET%
END SUB
```

Input parameters for locate-string:

SEARCH\$

Text to search for in 3270 PS

DIRECTION\$

Flag:

- | | |
|---|-----------------|
| 0 | Search forward |
| 1 | Search backward |

Return codes for locate-string:

RETURN.CODE%

- | | |
|----|--|
| 0 | String found. |
| -1 | Function did not complete successfully due to one of the following system errors: <ul style="list-style-type: none"> • No response from 3270 emulation • Error reading or writing pipe |
| -2 | API not connected. |
| -3 | Incorrect parameter specified. |
| -4 | Request rejected by 3270 emulation. |

Output parameters for locate-string:

OFFSET%

Offset of field within the PS. Specified as absolute offset within the PS (0 to 1919).

ATTRIB%

The field attribute byte is returned in exactly the form received from the host in the 3270 data stream.

Locate-field

The Locate-Field verb (S3.LOCFIELD) searches for the next field from current cursor position.

Syntax for locate-field: An example of the Locate-Field verb follows:

```
SUB S3.LOCFIELD(DIRECTION%, OFFSET%, ATTRIB%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
INTEGER*1 DIRECTION%
INTEGER*2 OFFSET%
INTEGER*1 ATTRIB%
END SUB
```

Input parameters for locate-field:

DIRECTION%

Flag:

- | | |
|---|-----------------|
| 0 | Search forward |
| 1 | Search backward |

Return codes for locate-field:

RETURN.CODE%

- | | |
|----|--|
| 0 | Field located successfully. |
| -1 | Function did not complete successfully due to one of the following system errors: <ul style="list-style-type: none"> • No response from 3270 emulation • Error reading or writing pipe |
| -2 | API not connected. |
| -3 | Incorrect parameter specified. |
| -4 | Request rejected by 3270 emulation. |

Output parameters for locate-field:

OFFSET%

Offset of field within the PS. Specified as absolute offset within the PS (0 to 1919).

ATTRIB%

Field attribute byte. The field attribute byte is returned in exactly the form received from the host in the 3270 data stream.

Locate-cursor

The Locate-Cursor verb (S3.LOCCURS) returns the offset of the cursor within the PS.

This function is needed because the cursor can be moved to specific points on the screen under host control. The application might need to know where the host has left the cursor and might need to send a different response depending on where the cursor is found.

Syntax for locate-cursor: An example of the Locate-Cursor verb follows:

```
SUB S3.LOCCURS(OFFSET%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
INTEGER*2 OFFSET%
END SUB
```

Input parameters for locate-cursor: There are no input parameters associated with the Locate-Cursor verb.

Return codes for locate-cursor:

RETURN.CODE%

- | | |
|----|---|
| 0 | Successful function. |
| -1 | Function did not complete successfully due to one of the following system errors: <ul style="list-style-type: none">• No response from 3270 emulation• Error reading or writing pipe |
| -2 | API not connected. |

Output parameters for locate-cursor:

OFFSET%

Offset of cursor within the PS. Specified as absolute offset within the PS (0 to 1919).

Set-cursor

The Set-Cursor verb (S3.SETCURSOR) sets the cursor to a specified offset in the PS.

Syntax for set-cursor: An example of the Set-Cursor verb follows:

```
SUB S3.SETCURSOR(OFFSET%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
INTEGER*2 OFFSET%
END SUB
```

Input parameters for set-cursor:

OFFSET%

New offset of cursor within the PS. The following is the absolute offset within the PS:

- | | |
|------|---------------------|
| 0 | Top left corner |
| 1919 | Bottom right corner |

Return codes for set-cursor:

RETURN.CODE%

- 0 Successful function.
- 1 Function did not complete successfully due to one of the following system errors:
 - No response from 3270 emulation
 - Error reading writing pipe
- 2 API not connected.
- 4 Request rejected by 3270 emulation.

Output parameters for set-cursor: There are no output parameters associated with the Set-Cursor verb.

Query-OGL

The Query-OGL verb (S3.QUERYOGL) returns information about the status of the operator guidance line (OGL) by specifying if a given message is present.

Table 49 contains valid message numbers and their associated OGL messages.

Note: The text in uppercase is the message that appears on the OGL when using U.S. English. Text in lowercase is descriptive.

Table 49. Valid message numbers for OGL messages

Message number	OGL message
1	FUNCTION UNAVAILABLE
2	PRINTER BUSY
3	PRINTER ERROR
4	PRINTING
5	KEY INPUT LOST
6	PROG <i>nnn</i>
7	FIELD FULL
8	PROTECTED
9	RANGE CHECK
10	COMM <i>nnn</i>
11	SYSTEM
12	WAIT
18	INSERT
19	NUMERIC
20	PRINTER <i>nn</i>
21	Blank message field
22	SSCP
23	Application name when session is bound
28	LU ID

Syntax for query-OGL: An example of the Query-OGL verb follows:

```
SUB S3.QUERYOGL(MESSAGE%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
INTEGER*1 MESSAGE%
END SUB
```


Input parameters for query-UGL:

MESSAGE%

Message number to query (values 1 to 28).

Return codes for query-UGL:

RETURN.CODE%

- 0** Message present.
- 1** Function did not complete successfully due to one of the following system errors:
 - No response from 3270 emulation
 - Error reading or writing pipe
- 2** API not connected.
- 3** Incorrect parameter specified.
- 4** Message not present.

Output parameters for query-UGL: There are no output parameters associated with the Query-UGL verb.

Send-key

The Send-Key verb (S3.SENDKEY) inputs a keystroke as from the keyboard.

For graphic keystrokes, the value of the KEY.CODE% parameter is as defined by the FLEXOS** 16-bit Output Character Set. See the *FLEXOS System Guide* for more information.

For example, to input the letter A, KEY.CODE% is set to 65 (X'41', the ASCII value of A).

Table 50 defines the values of the KEY.CODE% parameter for 3270 function keystrokes.

Table 50. KEY.CODE% Values for 3270 function keystrokes

KEYCODE% Value	Key	Function
4	SysRq	System Request
5	ATTN	Attention
6	RESET	Reset
8	UP	Cursor UP
9	DOWN	Cursor DOWN
10	RIGHT	Cursor RIGHT
11	LEFT	Cursor LEFT
12	RIGHT2	Cursor Right Double
13	LEFT2	Cursor Left Double
14	TABKEY	Cursor Field Tab
15	BACKTAB	Cursor Field Backtab
16	HOME	Cursor Home
17	NEWLINE	Cursor New Line
18	INSERT	Insert Mode
19	DELETE	Delete Character
20	Erase-IP	Erase Input
21	ERASE-EOF	Erase End of Field

Table 50. KEY.CODE% Values for 3270 function keystrokes (continued)

KEYCODE% Value	Key	Function
22	DUP	Duplicate
23	FLDMARK	Field Mark
26	ENTER	Enter
27	CLEARKEY	Clear
28	PA1	Program Access 1
29	PA2	Program Access 2
30	PA3	Program Access 3
31	PF1	Program Function 1
32	PF2	Program Function 2
⋮	⋮	⋮
54	PF24	Program Function 24
55	CURSEL	Cursor Select
56	PRINT	Local Copy Print
57	IDENT	Identify Local Copy Device
58	DEVCNCL	Device Cancel

Table 51 describes the values for device control key codes.

Table 51. Device control key codes

KEYCODE% Value	Key	Function
6	COLRMODE	Color mode toggle
7	NUMORIDE	Numeric override

The 3270 function key PRINT (Local-Copy) is a useful tool for debugging APIs.

Using the Send-Key function to enter the Local-Copy key causes a printout of what would be displayed on the screen if an operator, rather than a program, was using the application. After issuing the Local-Copy, the API should issue a Wait verb to ensure that the keyboard is unlocked. The keyboard is locked while the printing is in progress.

Note: The Local-Copy print function for 3270 emulation, unlike the Print Screen for the operating system, does not print an exact duplicate of the screen image. See the *4690 OS: User's Guide* for details.

Syntax for send-key: An example of the Send-Key verb follows:

```
SUB S3.SENDKEY(KEY.CODE%, KEY.TYPE%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
INTEGER*1 KEY.CODE%
INTEGER*1 KEY.TYPE%
END SUB
```

Input parameters for send-key:

KEY.CODE%
Keystroke value.

KEY.TYPE%

Flag:

- | | |
|---|--------------------------|
| 1 | Graphic keystroke |
| 2 | 3270 function keystroke |
| 3 | Device control keystroke |

Return codes for send-key:**RETURN.CODE%**

- | | |
|----|--|
| 0 | Successful function. |
| -1 | Function did not complete successfully due to one of the following system errors: <ul style="list-style-type: none"> • No response from 3270 emulation • Error reading or writing pipe |
| -2 | API not connected. |
| -3 | Incorrect parameter specified. |
| -4 | Request rejected by 3270 emulation (incorrect keystroke). |

Output parameters for send-key: There are no output parameters associated with the Send-Key verb.

Send-string

The Send-String verb (S3.SENDSTRING) inputs a string of graphic characters to the emulator.

The function of Send-String can be achieved by a series of Send-Key verbs.

If the string does not fit in the field available, the emulator sends a RETURN.CODE% of -5. The string has not been accepted by 3270, meaning the original contents of the field remain. The LENGTH% parameter returned tells the application the maximum length of string that can be accepted.

Syntax for send-string: An example of the Send-String verb follows:

```
SUB S3.SENDSTRING(IN.STRING$, LENGTH%, RETURN CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
STRING IN.STRING$
INTEGER*2 LENGTH%
END SUB
```

Input parameters for send-string:**IN.STRING\$**

Character string to send.

Return codes for send-string:**RETURN.CODE%**

- | | |
|----|--|
| 0 | Successful function. |
| -1 | Function did not complete successfully due to one of the following system errors: <ul style="list-style-type: none"> • No response from 3270 emulation • Error reading or writing pipe |
| -2 | API not connected. |
| -3 | Incorrect parameter specified. |
| -4 | Request rejected by 3270 emulation. |
| -5 | Input string truncated. |

Output parameters for send-string:

LENGTH%

Length of string actually entered.

Copy-string

The Copy-String verb (S3.COPYSTRING) copies a string to the API data space.

This function is used for extracting data sent from the host. In effect, it simulates the operator reading data from the screen.

Syntax for copy-string: An example of the Copy-String verb follows:

```
SUB S3.COPYSTRING(OUT.STRING$, OFFSET%, LENGTH%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
STRING OUT.STRING$
INTEGER*2 OFFSET%
INTEGER*2 LENGTH%
END SUB
```

Input parameters for copy-string:

OFFSET%

Offset in the PS from which to copy. Specified as the absolute offset within the PS:

0 Top left corner

1919 Bottom right corner

LENGTH%

Length of data to copy.

Return codes for copy-string:

RETURN.CODE%

0 Successful function.

-1 Function completed unsuccessfully due to one of the following system errors:

- No response from 3270 emulation
- Error reading or writing pipe

-2 API not connected.

-3 Incorrect parameter specified.

-4 Request rejected by 3270 emulation.

Output parameters for copy-string:

OUT.STRING\$

Data string extracted.

LENGTH%

Length of data copied. Copying terminates if the end of the PS buffer is encountered.

Copy-field

The Copy-Field verb (S3.COPYFIELD) copies the contents of the field at the given offset.

Syntax for Copy-Field: An example of the Copy-Field verb follows:

```
SUB S3.COPYFIELD(OUT.FIELD$, OFFSET%, LENGTH%, RETURN.CODE%) EXTERNAL
INTEGER*2 RETURN.CODE%
STRING OUT.FIELD$
INTEGER*2 OFFSET%
INTEGER*2 LENGTH%
END SUB
```

Input Parameters for Copy-Field:

OFFSET%

Offset in the PS of the field to copy.

LENGTH%

Length of data to copy.

Return Codes for Copy-Field:

RETURN.CODE%

- | | |
|-----------|---|
| 0 | Successful function. |
| -1 | Function did not complete successfully due to one of the following system errors: <ul style="list-style-type: none">• No response from 3270 emulation• Error reading or writing pipe |
| -2 | API not connected. |
| -3 | Incorrect parameter specified. |
| -4 | Request rejected by 3270 emulation. |

Output Parameters for Copy-Field:

OUT.FIELD\$

Data string extracted.

LENGTH%

Length of data copied. Copying terminates if the end of the PS is encountered. Fields can wrap at the end of the screen. COPY-FIELD returns the data between the cursor and the end-of-screen if this occurs in the current field.

Writing application programs for use with the 3270 API

When writing an application using the 3270 API, you have to know what will appear on the 3270 emulation screen in all conditions. The API depends completely on the characteristics of the host application with which the interface is communicating.

For example, if the host application is a customer database that prompts the 3270 operator to enter CUSTOMER NAME:, ADDRESS STREET:, CITY:, STATE:,ZIP:, the application must:

1. Locate the string "CUSTOMER NAME" (Locate-String).
2. Move the cursor to the input field (Locate-Field).
3. Enter the customer name (Send-String "BILL SMITH").
4. Locate the String "ADDRESS STREET" (Locate-String).
5. Move the cursor to the input field (SET-CURSOR).
6. Enter the street (2903 Meadow Lane) (Send-String).
7. Enter the city (Send-String).
8. Enter the state (Send-String).
9. Enter the zip (Send-String).
10. Send the Enter keystroke (Send-Key).

You can design the application in other ways. If you know where the first input field on the PS is located, you can write the code for input to the following fields:

1. SET-CURSOR to start of name input field
2. SEND-STRING (customer name)
3. SET-CURSOR to start of street input field
4. SEND-STRING (street)
5. ... (and so on).

Or, if you know the order of the fields on the screen then the NAME field is the third field, the street field is the seventh, and so on. For example:

1. SET-CURSOR 1 (to ensure the cursor starts at top left)
2. LOCATE-FIELD (finds the first field on the screen)
3. LOCATE-FIELD
4. LOCATE-FIELD (finds the third field ... (CUSTOMER NAME input))
5. SEND-STRING (customer name)
6. ... (and so on).

If the host application and the 3270 API get out of step, the results are unlikely to be useful. The application will be searching for strings that do not appear in the PS. An operator can usually react to this problem and recover by using help information or by starting again. Only a very sophisticated 3270 API application would be able to recover without operator assistance.

In developing the application, the following suggestions can be helpful:

- Put tracing code in the application while it is under development.
- The API can carry out 3270 operations only in the same way as an operator. It is essential that the application processes be tested manually. This helps you to determine the sequence of the screens and the keys to be entered by the application.
- Use the Send-Key verb (Local-Copy) to print screen images periodically while the application is being debugged. It might be necessary to issue a Wait verb to ensure that printing is complete before entering data.

Some 3270 host application sequences might be time dependent. If a Local-Copy is used during a time-dependent operation, it might delay the host application, so that it acts differently when the Local-Copy is not used. The application, when debugged using the local copy prints, fails when the prints are removed.

- The application can be developed on the store controller and ported easily to the terminal later. This is easier than developing the application on the terminal because the store controller offers a better development environment such as multiple windows, debugging tools, and editors.
- Use non-API sessions with actual screen and keyboard I/O to test assumptions about the functions of the host application.
- The 4690 application can be written to work with a host application that has been custom-written for the API. In this case, the 4690 application programmer and the host application programmer should be able to work together to make the job simpler. If the host application is not intended to be used by an operator, it does not need a user-friendly screen layout or help screens. It can be written to be program-friendly instead.
- Check for error conditions while you are developing the application. For example, if data is accidentally entered while the keyboard is locked, then a Function Unavailable error appears that you must clear by pressing **Reset**. It is recommended that a Query-OGL verb be issued to check for Function Unavailable and a Reset sent before attempting to send additional keys. If not corrected, the next API verb times out. If an API verb times out, the API verb considers the connection to the emulator to be lost. The application cannot issue an S3.DISCONNECT to stop the emulator. If it was started in the background, the emulator has to be stopped from the background application control screen. For this reason, the application should start the emulator using the S3.CONNECT call with a unique message to be displayed on the background screen.

Compiling and linking

The file ADXVB3AI.J86 contains the external program definitions for the API verbs. This file should be included in the compilation units of the application using the API application. A *compilation unit* is every module that makes a call to the API.

The ADXVB3AI.J86 file is used for all memory models and environments, including large memory model/controllers, large memory model/terminals, and medium memory model/terminals.

When the application is linked, one of the following libraries must be included in the link list:

ADXHS3LL.L86	For store controller applications
ADXZE3LL.L86	For large memory model terminal applications
ADXZE3ML.L86	For medium memory model terminal applications

The API verbs make use of three file I/O session numbers. The allocation of the session numbers is left to the application programmer so that the same verbs can support many user applications. If the application does not set them, defaults of 31, 32, and 33 are used. These defaults are suitable for 4680 General Sales Application (GSA) user exits but not necessarily for any other application.

To set the file numbers to be used, the application programmer must:

1. Define three file I/O session numbers not used by the terminal application or any user exit code. If Toshiba-supplied applications are being used, consult the session number reference in the associated programming guide.
2. Include the following global declarations with the program variable declarations:

```
INTEGER*2 GLOBAL S3.INBOUND
INTEGER*2 GLOBAL S3.OUTBOUND
INTEGER*2 GLOBAL S3.SEMAPHORE
```
3. In initialization code, assign one of the reserved session numbers to each of these variables. The assignments must be made before S3.CONNECT is called.

Keyboard and language combinations

This section gives information on the National Language Support (NLS) approach taken within the 3270 emulator and its use of the operating system ASCII-to-EBCDIC translation functions, the NLS file structure and contents, and an example of how to reconfigure the file for a different language.

This information is required reading for anyone attempting to prepare a new language variant of the ADXHS30F.DAT file. The sections on keyboard support are especially important.

ADXHS30F.DAT is the NLS file for 3270 emulation. NLS for 3270 emulation consists of several versions of the ADXHS30F.DAT file. Each file has the messages translated and the ASCII-to-EBCDIC keycode conversions for the country and language/keycode combinations listed in Table 52. You can use the Supplemental Diskettes or the Supplemental Option using the CD-ROM to select the country and language/keycode you need.

Table 52. National Language Support for 3270 emulation

Country (keyboard)	Language/keycode
U.S.	English/U.S.
U.K.	English/U.K.
Canada	Canadian-French/Canadian
France	French/French
Germany	German/German
Spain	Spanish/Spanish
Norway	Norwegian/Norwegian (See note 1)
Denmark	Danish/Danish (See note 1)
Italy	English/Italian (See note 2)
Portugal	Portuguese/Portuguese (See note 1)

Note:

1. The ANPOS keyboard is not available in this language.
2. The 3270 messages are not available in Italian.

The installation process selects the correct file based on your choice of keyboard. The choice is made according to the best match. For example, if you select the German keyboard, the German 3270 emulation file is selected. If you select the Switzerland-German keyboard, because there is no Switzerland-German 3270 emulation file, the German 3270 emulation is selected. You can alter the selected ADXHS30F.DAT file to take into account the differences between the German and Swiss-German ASCII-to-EBCDIC code conversions.

The installation process selects the 3270 emulation file for the remaining keyboards as illustrated in Table 53.

Table 53. National Language Support for countries without 3270 emulation files

Country (keyboard)	Language/keycode
Belgium	French/French
Brazil	English/U.S.
Greece	English/U.S.
Hungary	English/U.S.
Netherlands	English/U.K.
Sweden/Finland	English/U.S.
Switzerland-German	German/German
Switzerland-French	French/French
Switzerland-Italian	English/Italian
Turkey	English/U.S.

ASCII-EBCDIC translation functions

A 3270 emulation session receives EBCDIC-coded data from the host for display on the screen. This EBCDIC-coded data must be converted to ASCII-coded data. A 3270 emulation session receives ASCII-coded data from the screen, the keyboard, or the APL for transmission to the host. This ASCII-coded data must be converted to EBCDIC.

These conversions are performed using the translation table that is used by the operating system. The translation table holds the 256 code points for the language you select. These code points correspond to the 256 possible EBCDIC code points.

National Language variations in the EBCDIC character sets require alterations to the translation table. The operating system ADXSERVE function initializes this table using the language variant installed in the operating system as the default. For example, to use a Spanish variant of the 3270 emulation, the Spanish version of the operating system must be installed.

During controller initialization, ADXSERVE is used to initialize the EBCDIC-to-ASCII and the keycode-to-EBCDIC-character code conversion tables.

For terminal emulations, the ADXSERVE function is not available. Terminal emulations use two controller programs that enable them to use the SNA driver. One of these, during its initialization, constructs an EBCDIC-to-ASCII character code conversion table and writes it into memory. Each terminal emulation reads this file as it initializes.

NLS file structure and contents

Initialization of 3270 emulation uses the single NLS file for all instances of emulation in a system of controller and terminals. The file is in ASCII format and can be altered with a PC editor. It contains six sections, one for each area affected by NLS. Each section is preceded by instructions for making alterations.

- Keycode Mapping
 - (SECTION 1) Controller
 - (SECTION 2) Auxiliary console
 - (SECTION 3) Terminal
- Messages
 - (SECTION 4) OGL
 - (SECTION 5) Background Status
 - (SECTION 6) API

The data in the file must appear in the correct order. A 3270 emulation session cannot detect the error if Background Status messages appear where OGL messages are expected.

Any data following an asterisk (*) is regarded as a comment and is ignored by the 3270 emulation until an end of line is reached. If an asterisk is required in the OGL message, for example, it must be preceded by a backslash (\) character.

Keycode mapping

Different keyboards attached to controllers and terminals are supported. The emulation program receives a code from the device driver to indicate which key was pressed. This code is then translated into an EBCDIC code point.

Variation in the National Language requires certain modifications in the keycode-to-EBCDIC-code translation table. National language variants of the EBCDIC codes might require different graphic characters to be associated with the same keycode.

National Language considerations affect only keys that correspond to graphic characters. National Language considerations do not affect function keys, such as the cursor movement keys.

Some graphic character keys on a real 3270 keyboard have no equivalent key on the standard controller keyboard, the remote keyboard, or the ANPOS or 4693 ANPOS terminal keyboard. The operator must be able to type all of the characters that can be keyed into a 3270 87-key keyboard.

The solution is to remap the unique PC, ANPOS, or 4693 ANPOS keys to produce the unique 3270 keyboard graphic characters. For example, an English (U.S.) 3270 keyboard has keys for the cent monetary symbol (¢) and the NOT symbol (¬). It does not have keys for the left bracket ([) or caret (^) symbols. Therefore, for a 3270 emulation configured with English (US) as the National Language, the [key is mapped to the ¢ monetary symbol, and the ^ key is mapped to the ¬ symbol.

A table in the NLS file performs the mapping. The emulator reads in this table and modifies the ASCII-to-EBCDIC translation table accordingly. The table specifies the ASCII keycode (in hexadecimal notation) of the keyboard key, and the EBCDIC code to use when this key is pressed.

This is not a straight translation. The ASCII value for [(the left bracket symbol) is not translated to the EBCDIC value for [(the left bracket symbol). Instead, it is translated to the EBCDIC value for ¢ (the cent symbol).

Controller and Auxiliary Console Support: The following keyboards are supported:

- 3151 keyboard
- 3161 keyboard
- 3164 keyboard
- ANPOS
- 4693 ANPOS
- Modular ANPOS keyboard
- Modular 67-Key keyboard
- Modular 67-Key LCD keyboard
- PS/2® keyboard

The interfaces presented by the drivers that support these keyboards are all the same in that the format of data received by the emulation is identical. When a key is pressed, the 3270 emulation receives two bytes of information from the device driver. The first byte determines the type of key that has been pressed: a graphic key or a control character key. The second byte indicates the specific key that has been pressed.

The device driver provides the ASCII code for a graphic character. In most cases, the equivalent EBCDIC character code is obtained by using the ASCII-to-EBCDIC translation table built using ADXSERVE.

The NLS file contains two remapping tables for the controller emulation. SECTION 1 is used for the system console, and SECTION 2 is used for the auxiliary console. The 3270 emulator determines during initialization which type of console it is being run from and which table to use to remap its translate tables.

Terminal Support: The 3270 emulation supports the ANPOS and 4693 ANPOS keyboards on the terminal. The device driver for the ANPOS and 4693 ANPOS keyboards provides a 1-byte keycode to an application. This 1-byte code is translated into an EBCDIC code point or a 3270 emulation control keycode.

For graphic characters such as the QWERTY keys, the keycode received from the driver is the ASCII value of the keytop. This is translated using the ASCII-to-EBCDIC table. Certain keycodes are used to mean 3270 function or control keys. These codes are fixed and are not NLS dependent.

SECTION 3 in the NLS file is the remapping table for the terminal emulation.

Messages

Table 54 describes the information messages displayed by the 3270 emulation program.

Table 54. Information Messages Displayed by 3270 Emulation Program

Message Type	Section of NLS File	Maximum Length	Description
OGL	SECTION 4	80 characters	The OGL is a single line displayed at the bottom of the screen. It is divided into a number of fields for displaying messages of different types. Because the OGL is limited to 80 characters, the length of all of these fields is limited. OGL messages are held internally in an array that is initialized from the NLS file. OGL messages are arranged by type with the maximum field length of each type indicated.
Background Status	SECTION 5	40 characters	The 3270 emulation produces background status messages when running in the background. Background messages are held internally in an array that is initialized from the NLS file. There are four messages with a total of nine words.
API	SECTION 6	40 characters	<p>The 3270 emulation program produces API messages when:</p> <ul style="list-style-type: none"> • An application gains control of a 3270 LU 2 session • An application relinquishes control of a 3270 LU 2 session <p>API messages are held internally in an array that is initialized from the NLS file. There are two messages with a total of six words.</p>

How to Reconfigure the NLS File

Use the following general procedure for a National Language variant:

1. Determine which 3270 graphic characters in that language are not present on the 4690 keyboards (ANPOS, 4693 ANPOS, controller, and remote console) for that language.
2. Determine which 4690 keys are present that are not needed for the 3270 emulator (for example, the [symbol). The following are sources of the information needed to reconfigure the NLS file:

Keyboard	Reference
3270 Keyboards	<i>3174 Character Set Reference (GA27-3831)</i>
4680 Controller Keyboards	DOS manuals provided with the PS/2 system
3151/52/61/64 Keyboards	<i>3151 ASCII Display Station Reference Guide (GA18-2634)</i>
ANPOS Keyboard	<i>4690 OS: Planning, Installation, and Configuration Guide</i>
4693 ANPOS Keyboard	<i>4690 OS: Planning, Installation, and Configuration Guide</i>

3. Edit the remapping tables (the first three tables in the NLS file) to map unused ASCII keycodes to the 3270 EBCDIC codes that are needed.

For example, to support a French-Canadian language host:

- “Backquote” appears on the French 3270 typewriter keyboard but not on the French PS/2 keyboard.
- Backslash (\) appears on the French PS/2 keyboard but not on the French 3270 typewriter keyboard.

Other keys appear on the French PS/2 keyboard but not on the French 3270 typewriter keyboard.

However, the requirement is to be able to reproduce all keystrokes that are possible on a 3270 keyboard.

The changes made to the table are as follows:

Before:

```
*****
* ASCII  EBCDIC
* keycode code
*****
5B      4A * pressing the '[' key maps to 'cent monetary symbol'
5D      4F * pressing the ']' key maps to 'unbroken vertical bar'
5E      5F * pressing the '^' key maps to 'NOT symbol'
          * end of key mappings
```

After:

```
*****
* ASCII  EBCDIC
* keycode code
*****
5C      A0 * '\' maps to 'backquote'
          * end of key mappings
```

Chapter 10. Designing LU 0/SNA Programs

This chapter shows you how to design LU 0/SNA programs.

Link and Session Considerations

A single subarea link to the host can support multiple logical unit (LU) sessions and be simultaneously used by multiple LU 0 applications. Each application must have an individual LU 0 session. No two applications can share the same session. A single application can use multiple LU 0 sessions on the host link.

Valid session-level requests are open, close, read, write, and status. Valid link-level requests are open, close, and status.

Line Communications Protocol

SNA support for LU 0 applications can use SDLC or X.25 as its line protocol. An application does not know which line protocol is being used nor is any unique code required to support one protocol or the other. The application must specify only the link configuration name when the link is opened. The actual line protocol to be used is defined during the configuration process and related to the link configuration. Use care in defining your line configuration so it matches the physical connection you are using. For details about configuration, see the *4690 OS: Planning, Installation, and Configuration Guide*.

SNA Support for LU 0

This section contains information about the different kinds of support that are available for LU 0.

Transmission Control Support

The transmission control layer supports only Format Identification (FID) type 2 headers and does not provide segmenting support. The transmission control layer performs pacing of requests and responses from the host and performs sequence checks on normal flow requests received from the host. The transmission control layer also generates sequence numbers for application requests written to the host. In the case of responses, the application must verify and generate all sequence numbers. The transmission control layer does not provide data encryption support.

Data Flow Control Support

The application must implement and support the data flow control layer. The application must also provide support for the BIND request, the CLEAR request, and the Set and Test Sequence Number (STSN) SNA request. The SNA support passes these requests to the application and acts upon the response from the application. The SNA services support also passes cryptography verification (CRV) to the application and sends the response from the application to the host. Only EBCDIC data is supported. The application must perform any required conversions or translations.

When an STSN request is received, the application should modify only the RH fields to be a positive or negative response. The SNA support then processes that response and sets the RU fields of the STSN response as appropriate.

BIND Processing

The application must receive the BIND RU from the host system, verify the parameters that affect the application, and write the response RU.

When the SNA support receives the BIND response, it allocates the read and write buffers and the pacing queues for the session. If a negotiable BIND is being used, the SNA support reduces the number of

buffers allocated to support pacing if all of the storage cannot be obtained. As long as two read buffers and one write buffer can be allocated, the SNA support modifies the BIND response and transmits it to the host. If the storage is not available, the BIND response is not transmitted to the host. The application receives a memory error return code from the write request.

The application can then change the positive response for the BIND to a negative response and write it to the host, or it can return a positive response. The application cannot change the maximum RU size for non-negotiable BINDS.

SNA does not support negotiable BINDs for LU0. SNA rejects this type of BIND with sense code 08210000 without sending it to the application.

In the case of a non-negotiable BIND, if storage cannot be allocated or the maximum RU size exceeds 512 bytes, the SNA support changes the BIND response into a negative response and transmits the response to the host. The application receives a BIND error code in response to its write request for the BIND response. HCP requires an RU size of 256 bytes and RCMS requires an RU size of 256 bytes or 512 bytes. The size of the RU for other applications should be chosen carefully, because large RU sizes combined with large pacing values can quickly deplete SNA buffers, and SNA communication stops with Message W857 RC=80BC0B42.

Starting LU 0 Applications

Host applications in the store controller can be started from the operating system command sequence, from another application, or by SNA support through an unsolicited BIND request from an active host link.

Application Started by an Unsolicited BIND Request

When an SNA link exists between a store controller and a host, an SNA BIND request can flow from the host to the store controller. When the BIND is received, a host application is requesting that a store controller application be started. The name of the store controller application can be defined in the user data field of the SNA BIND request. If it is not specified there, the local application name specified in the configuration of a session for a host link is used. This name can be the actual name of an application as it resides on the store controller, or it can be a name that is expanded by the system to be the actual application name that is loaded.

The operating system provides an application to use as the local application if you are starting applications with the 3650 version of the ADCS Start User Program (SUP) command. The program name is ADXHSNPL286. Specify this name for the local application if you are using SUP through ADCS configured for a 3650. Do not specify this name as a permanent background application.

If the BIND request sent by the host contains the name of the store controller application to be started, it must be the first entry in the user data field. The first character of the name cannot contain a blank or be a null (0) character. The name of the program to be started can have a maximum length of 24 characters. The last character in the name must be either a null character or a X'01'. This character is for control purposes only and is not used as part the name when the command is given to the operating system to start the program.

A X'01' control byte in the application name indicates that the application will not enter into a dialog with the host system over the session on which the BIND request has been received. After the operating system receives the command to start the specified application, SNA support sends a BIND response to the host. If the operating system detects an error in the start program command, SNA support sends a negative response with sense code X'081C'. Otherwise, SNA support sends a positive response.

A null control byte in the application name indicates that the application being requested is to enter into a dialog with the host system. In this case, the BIND request must be processed by the application that gets started.

Regardless of what type of application is started, a parameter is always passed to it. The parameter contains the name of the SNA link configuration on which the BIND request flowed, a comma, and the name of the session configuration on which the BIND request was received. If a dialog is to be established with the host, the application must issue an OPEN request for the link and pass the session names in this parameter. If this is not done, the session on which the BIND was received might become unusable. The session will definitely become unusable if the application fails to respond to the BIND request after opening the link and the session.

When SNA support sends a positive BIND response to the host, it indicates that the operating system accepts the command to start the requested application. The operating system might not properly process the program, or the program could fail after being started. You must provide a means to verify whether the application processed successfully or not.

START USER PROGRAM (SUP) Command

The ADCS START USER PROGRAM (SUP) command is implemented as an unsolicited BIND request for an SNA session. The operating system supports two formats of this command:

- In the first format, the name of the program to be started follows the Function Management (FM) header. This is necessary for compatibility with the IBM 3650 system.
- In the second format, the name of the program to be started is in the user data portion of an SNA unsolicited BIND. This format is necessary for compatibility with the IBM 3680 system.

Opening an SNA Subarea Link

Before an SNA subarea link can be opened, it must first be defined using the configuration services. A system services control point (SSCP) ID is specified that must match the ID that appears in the ACTPU command received from the host when the link is activated. If the two IDs do not match, the store controller SNA support rejects the ACTPU request from the host and waits for another ACTPU or a disconnect request.

When a program requests an SNA link to be opened for the first time, there must be enough system memory available to load the communications code (if it has not been loaded). The communications code supports the line protocol to be used, the SNA services support, and all the required buffers and control blocks. As LU 0 sessions are opened, additional memory is required for session status and control.

The operating system provides an application to start an SNA link. The name of this application is ADXHSNLL.286 and is in the system program subdirectory, ADX_SPGM. You can also start an SNA link at the COMMUNICATIONS FUNCTION panel and by issuing an Enable Link command. See the *4690 OS: User's Guide* for more information on accessing the COMMUNICATIONS FUNCTION panel.

Opening an LU 0 Session

Before SNA sessions can be opened, they must be defined for the SNA link on which they operate. The definition is specified by the LU 0 session group configuration. The host program name that is defined in the session configuration name is the name that is sent to the host with an INIT SELF command. An INIT SELF command is automatically sent when the session is opened if a data area to receive the BIND has been specified in the open session request and no BIND has been received from the host system. See "OPEN SESSION" on page 212 for more information.

Closing LU 0 Sessions and Subarea Links

When an LU 0 session is closed by an application, the system resources that were required to support the session are freed, and the 4690-LU to host-LU session is ended. The SSCP to LU session remains active until the host issues a DACTLU request, or the link is disabled.

If an application closes a session and an UNBIND request has not been received from the host system, SNA support automatically sends a TERM SELF request to the host system and processes the responses from the host.

When the host system sends an UNBIND request and the store application is still active, the SNA support automatically sends a response to the host. The store application receives a return code from any session request to the SNA support stating that an UNBIND has been received. When an application receives this return code, the only valid request that can be made to the SNA support is a CLOSE request for the session. The SNA support rejects all other requests.

When a subarea link is closed, two factors determine if the system resources are freed:

1. If the link is opened by another application or the link is resident, the resources are not freed and the link remains enabled. The host SSCP to LU session remains active.
2. If no other applications have the link opened and the link used to support the link is not resident, then all system resources are freed, and the memory where the communications code was loaded is returned to the Operating System.

If a condition occurs that disables the link (such as a DACTPU request from the host or an unrecoverable error), the applications that have opened the link receive an error. These applications must end all sessions on the link and close the link.

When an ACTPU, ACTLU, DACTLU, or DACTPU request is received from the host, any store applications that have the link or sessions opened must close all the sessions and the link. The application can then reissue the OPEN commands for the link and the sessions.

4680 BASIC Statements for LU 0 Applications

The following section describes the guidelines for using 4680 BASIC statements when writing host communications applications.

OPEN LINK SNA

Use the OPEN LINK SNA statement to activate or gain access to the SNA host communications. The *comm.name* and *link.number* parameters are required. The *comm.name* parameter is the name of an SNA link as defined during configuration. Any number of different programs can open the same SNA link. Because many SNA links can be defined at configuration, care must be taken not to have two programs opening different links that require the same line. If this happens, an error code is returned indicating that the line is already in use.

Control is not returned to the user program until the host link has been successfully established and a valid ACTPU has been received. If an error occurs during the host link startup, an error code is returned to the program. If an ACTPU is received that is not valid, the system sends a negative response to the host system and waits for a valid ACTPU to be sent or for the host line to be disconnected.

The *link.number* parameter is a unique number used to identify the SNA link within an application program. This number must be used when a host session is opened or when the user program wants to obtain the status of the host link through a GETLONG function request. This number must be unique within the user program.

The *host.id* parameter is an optional parameter that can be specified on the OPEN LINK statement. This value is a maximum of 8 characters long and replaces the XID (controller ID) defined in the SDLC configuration. The parameter is a character representation of the hexadecimal XID value. This parameter is useful when testing new links or when used in a multipoint environment.

OPEN SESSION

Use the OPEN SESSION statement to start a host communications session. This session defines a logical link between the user's program and a program that resides in the host. At the time the OPEN request is

processed, an INIT SELF request is issued to the HOST system if a BIND has not been received and the link has been activated. If the link has not been activated, the INIT SELF is issued as soon as the link becomes active.

The *session.name* parameter is the name of a session that was defined for the previously opened link during the configuration process for host communications. A session can be established with only one application in the store controller at any one time. If an application in the host needs to communicate with multiple applications in the store controller, either it has to establish a session with each application, or it has to funnel data for all applications to one store controller application. The store controller application then passes the data to the appropriate store controller application.

The *link.number* parameter is needed for the correlation of session to link. See “OPEN LINK SNA” on page 212 for information about the *link.number* parameter.

The *bind.var* parameter is the name of a string variable in which the BIND request is stored coming from the host.

If *bind.var* has a length of other than zero (0), the user program receives control back after the BIND request has been received from the host system. The BIND request is placed in the *bind.var* buffer for processing by the user program. See “READ” to see how the data appears in the user program buffer area. If a *bind.var* parameter is used, it must be large enough to hold the BIND request sent from the host system plus the control information appended by the store controller SNA support.

If *bind.var* has a length of zero (0), control is returned to the user program as soon as the control blocks for the session have been obtained and initialized. The user program must then decide how to proceed. If the application at the host end is started from an action at the host, through a program or operator request, the store controller application can issue a READ request for the session that it just opened. This READ request is satisfied when the BIND request is received from the host system.

Alternately, the store controller application can request that the host system start the application that it wants to communicate with. This is done by issuing a PUTLONG session request to the SNA services with the INIT SELF flag set to 1. When control is returned to the store controller application from the PUTLONG request, the program can issue the READ request to obtain the BIND information to be sent by the host system.

Note: The PUTLONG request can only be issued after the host system has sent an ACTLU request to the store controller indicating that a session can be established with the host. The GETLONG statement can be used to determine if an ACTLU request has been received.

The *session.number* parameter is the number assigned to the session when it is opened.

READ

Use the READ statement to obtain session data sent from the host. The read buffer must be large enough to contain the RU size of the host request or response, plus a 3-byte RH field, plus eight bytes of control information. The maximum RU size that can be supported is 512 bytes. The *session.number* is the number assigned to the session when it is opened.

The data received from the host is placed into the *string.var* data area. The data appears exactly as it is sent from the host system. No conversion to ASCII characters is performed. The data stored in *string.var* is organized as follows:

Size (in Bytes)

Description

- | | |
|---|---|
| 4 | The size in bytes of the RU field. |
| 1 | The address of the host program sending the data. |

- 1 A 0 or a 1 to signify if the data was sent expedited:
 - 0 – not expedited
 - 1 – expedited
- 2 The sequence number of the data received from the host. The sequence number should be used to send any response to the host if this is a request.
- 3 The RH field of the data sent from the host.
- N The RU field of the data received from the host. The size of this field is contained in the first four bytes of this buffer.

Refer to the appropriate SNA guide for descriptions of the data that is stored in the RH field and or the non-user data that might be in the RU field. Examples of non-user data are:

- CLEAR requests
- Cryptography type requests
- Data flow control responses
- Data flow control requests
- STSN requests

The user program must ensure that its *string.var* area is large enough to hold a complete request or response sent by the host. A data overrun error code is returned if it is not.

GETLONG

Use this statement to obtain the status of SNA communications. You can use this status to determine what action to take. You should always obtain session status before requesting the SNA support to wait on some status change. For example, an application has started a session and requests an INIT SELF be transmitted to the host. The application then issues a WAIT command. The WAIT command is satisfied when acknowledgment of the INIT SELF request is received or if an error occurs in the transmission of the INIT SELF request to the host. To determine this, use the GETLONG command to check the status bits that are returned.

i4 is a 4-byte integer that contains status information about the link or session. This status information is described in the *4680 BASIC: Language Reference*.

WRITE FORM

Use the WRITE FORM statement to transmit data to the host. When transmitting data to the host system, the same requirements and limitations apply to the write as applied to the read buffer as previously mentioned, except for the maximum size of the RU field. The total number of bytes that can be transmitted to the host is 518. Because 11 bytes of the control information precede the RU field in *string.var*, the RU field on a WRITE FORM statement cannot exceed 507 bytes. The syntax for WRITE FORM is described in the *4680 Basic Language Reference*.

A write transaction returns control when the data has been placed in a transmit buffer or if an error has been detected. If an error code is returned, no data is transferred. If an error code is not returned, the number of bytes actually stored in the transmit buffer is returned. This does not imply that the host receives the data. Check the status of the SNA session and expected responses from requests to determine if all the data arrived at the host.

The *session.number* parameter is the number assigned when the session is opened. *String.var* must contain the same fields as the READ *string.var* data area. However, you need not specify the OAF address and the expedited indicator. Also, if you are sending a request to the host system, the SNA support generates the sequence number instead of picking it up from *string.var*. The user program must ensure that all the data in the RH and RU fields is specified correctly. This data includes the sequence number on any responses sent to host requests.

PUTLONG

Use the PUTLONG statement to tell the SNA support to send a request to the host and to process the response. Control is returned to the user program when the request is built and queued for transmission to the host. The requests that are supported are INIT SELF, TERM SELF, and RQR. Syntax for the PUTLONG statement is defined in the *4680 BASIC: Language Reference*.

RESUME

The RESUME statement cancels the operation that failed and allows the program to continue operating. If the RETRY parameter is specified, an OPEN is retried. RESUME RETRY for a read or write operation always returns an error code. The syntax for RESUME is described in the *4680 Basic Language Reference*.

CLOSE

The CLOSE statement ends a program's use of a session or link. The *number* parameter is the number that is assigned to the *session.number* or *link.number* used in the OPEN statement.

The last close of a link allows the system to remove SNA services code from memory once the host connection is broken if:

1. All communication links are configured with COMMUNICATIONS DRIVERS Resident=N and,
2. If all other host communication links are closed.

This memory is free to be used for other functions. If the link is subsequently OPENed again, SNA services are re-loaded into memory.

When the last close of a link is issued, an error is returned if any sessions remain open on the link.

The QUIESCE function is not implemented. The parameter is ignored.

The syntax for CLOSE is described in the *4680 Basic Language Reference*.

Example of an SNA Interface with the Host Application

The following example will assist you in writing a similar application to interface to SNA to communicate with a host application. The statement structure can be found in the *4680 BASIC: Language Reference*.

Depending on the protocols used by the operating system and host applications, the 4680 application might be required to handle session control and data flow control commands received from the host.

```
SUB ASYN.ERR(RETRY.FLAG,OVERLAY.STR$)
! Routine to handle async errors, such as write errors.
  INTEGER*2  RETRY.FLAG
  STRING    OVERLAY.STR$
! Code to handle asynchronous errors.
END SUB
! Any additional subroutines and functions needed by application to
! process data between the store controller and the host.
! Main line code - infinite loop to process the host after the
! link and session have been established, a BIND has been received and
! returned to the host and a SDT has been received from the host.
! Assumption is that communications with the host is via existing
! 4680 SNA drivers.
ON ASYNC ERROR CALL ASYN.ERR(RETRY.FLAG,OVERLAY.STR$)
ON ERROR GOTO SYN.ERR
OPEN SNA LINK link.name AS link.number ! link.name = name defined
! During configuration
IF OPEN LINK SUCCESSFUL THEN,

OPEN SESSION session.name ON link.number BINDVAR bind.area \
      session.number          ! Session.name = name defined
```

```

! During configuration,
! Link.number defined in OPEN SNA LINK.
! Bind.area is a string to hold the BIND information on completion of
! the OPEN SESSION. If bind.area is a null string, the OPEN SESSION
! completes when all control blocks have been allocated and the
! application must determine if the BIND has been received by
! the SNA driver. When the BIND has been received, the application
! must READ # the BIND and return it to the host before
! continuing processing.

WAIT SESSION.NUMBER
! Wait for host to send
I4 = GETLONG(SESSION.NUMBER)
! to determine if a BIND has been received from the host.
! If BIND received then, send bind back to the host.
WAIT SESSION.NUMBER
! Wait for host again
I4 = GETLONG(session.number)
! to determine if SDT was received.
! If SDT received then, communications established
! between application at 4690 and application at host.
! The preceding should not be executed again.
! Infinite loop.
WHILE A = A
! Infinite loop to process the host communications.
! Application is started by a unsolicited BIND
! from a host application.
    I4 = GETLONG(session.number)
! If write buffer available then
! use GETLONG to see if can write to host.
    WRITE FORM format.str$; # session.number; data.str$
! Wait for response from host
    WAIT session.number
    I4 = GETLONG(session.number)
! If data to be read, then
! receive response from host,
! process data received from host,
! prepare reply to host.
    READ# session.number; LINE host.resp.str$
WEND
SYN.ERR:
! Process synchronous errors - read, open, etc.
END

```

Chapter 11. Designing LU 6.2/SNA Programs

Support of LU 6.2 by the operating system provides for user-written transaction programs (TPs) on the 4690 store controller. The TPs have access to advanced program-to-program communications (APPC) functions and can communicate with APPC applications on adjacent SNA nodes. The TP accesses APPC functions through calls from a set of library routines that are linked to the TP.

The library routines provide an interface for TPs that are written in C, COBOL, or 4680 BASIC application program languages. The library files are located on the 4690 Optionals under the following names:

- ADXHSV0L.L86 (4680 BASIC)
- ADXHSV1L.L86 (COBOL)
- ADXHSV2L.L86 (C language)

The library routines support the call interface that is specified in the *Systems Application Architecture: Common Programming Interface: Communications Reference*.

CPI Communications Calls

New call subroutines conforming to CPI Communications specifications provide APPC capability that enables store controller applications to function as LU 6.2 transaction programs. Table 55 lists the calls supported by the operating system. See the *Systems Application Architecture: Common Programming Interface: Communications Reference* for the syntax and description of the CPI Communications calls.

Table 55. CPI Communications Calls

Pseudonym	Call
Accept_Conversation	CMACCP
Allocate_Conversation	CMALLC
Confirm	CMCFM
Confirmed	CMCFMD
Deallocate	CMDEAL
Extract_Conversation_Type	CMECT
Extract_Mode_Name	CMEMN
Extract_Partner_LU_Name	CMEPLN
Extract_Sync_Level	CMESL
Flush	CMFLUS
Initialize_Conversation	CMINIT
Prepare_To_Receive	CMPTR
Receive	CMRCV
Request_To_Send	CMRTS
Send_Data	CMSEND
Send_Error	CMSERR
Set_Conversation_Type	CMSCT
Set_Deallocate_Type	CMSDT
Set_Fill	CMSF
Set_Error_Direction	CMSED
Set_Log_Data	CMSLD
Set_Mode_Name	CMSMN

Table 55. *CPI Communications Calls (continued)*

Pseudonym	Call
Set_Partner_LU_Name	CMSPLN
Set_Prepare_To_Receive_Type	CMSPTR
Set_Receive_Type	CMSRT
Set_Return_Control	CMSRC
Set_Send_Type	CMSST
Set_Sync_Level	CMSSL
Set_TP_Name	CMSTPN
Test_Request_To_Send_Received	CMTRTS

4690 Extensions

The operating system provides a number of routines that are extensions to Systems Application Architecture® (SAA®) CPI Communications calls. Programs using these routines might not be portable to other SAA systems. See the *Systems Application Architecture: Common Programming Interface: Communications Reference* for the general syntax of the 4690 extensions.

Table 56 summarizes the 4690 routines that are extensions to CPI Communications. The routines are listed in alphabetical order by their call name. The last column of the table shows the page in this chapter where the routine is described in detail.

Table 56. *Overview of 4690 Extension Routines*

Call	Function	Parameters	Page
XCMCDL	Conditional disable of a link.	XCMCDL(<i>link_name</i> , <i>link_name_length</i> , <i>return_code</i>)	219
XCMDL	Force disable of link.	XCMDL(<i>link_name</i> , <i>link_name_length</i> , <i>return_code</i>)	219
XCMELE	Enable link function.	XCMELE(<i>link_name</i> , <i>link_name_length</i> , <i>return_code</i>)	220
XCMGLE	Get additional error information.	XCMGLE(<i>verb_id</i> , <i>CPIC_r_c</i> , <i>errcode1</i> , <i>errcode2</i>)	220
XCMQL	Query link status.	XCMQL(<i>link_name</i> , <i>link_name_length</i> , <i>return_code</i>)	224
XCMSTO	Set timeout value for CPI Communications functions.	XCMSTO(<i>conv_id</i> , <i>timeout_value</i> , <i>return_code</i>)	224

Table 57 on page 219 provides information about the type and length of each variable that can be used with 4690 extension routines.

Table 57. Variable Types and Lengths

Variable	Variable Type	Length
<i>CPIC_r_c</i>	Integer	32 bits
<i>errcode1</i>	Integer	32 bits
<i>errcode2</i>	Integer	32 bits
<i>link_name</i>	Character string	1 - 8 bytes
<i>link_name_length</i>	Integer	32 bits
<i>timeout_value</i>	Integer	32 bits
<i>verb_id</i>	Integer	32 bits
<i>return_code</i>	Integer	32 bits

Conditional Disable Link (XCMCDL)

Use the Conditional Disable Link routine (XCMCDL) in a program to request that a communications link be disabled when no applications are using the link. When XCMCDL is issued, another application cannot use the link until the link is enabled again.

Note: SDLC and X.25 links are supported by XCMCDL. Local link and token ring are not supported.

Format

```
CALL XCMCDL(link_name,
            link_name_length,
            return_code)
```

Parameter

Description

link_name

Specifies the name of the subarea link configuration record or the peer link configuration record.

link_name_length

Specifies the number of characters in the *link_name*.

return_code

Specifies the library routine return code.

Disable Link (XCMDL)

Use the Disable Link routine (XCMDL) in a program to request that the communications link be disabled unconditionally. Active applications must close the link or end the conversation for XCMDL to complete. Any active applications on the link following the initiation of XCMDL receive errors for any attempt to use the link.

Format

```
CALL XCMDL(link_name,
            link_name_length,
            return_code)
```

Parameter

Description

link_name

Specifies the name of the subarea link configuration record, the peer link configuration record, or the reserved name. The reserved name for the local link is ADXLOCAL

link_name_length

Specifies the number of characters in the *link_name*.

return_code

Specifies the library routine return code.

Enable Link (XC MEL)

Use the Enable Link routine (XC MEL) to enable a link for communications.

Format

```
CALL XC MEL(link_name,
            link_name_length,
            return_code)
```

Parameter**Description****link_name**

Specifies the name of the subarea link configuration record, the peer link configuration record, or the reserved name. A reserved name for the local link is ADXLOCAL.

link_name_length

Specifies the number of characters in the *link_name*.

return_code

Specifies the library routine return code.

Get Last Error (XCMGLE)

Use the Get Last Error routine (XCMGLE) to extract error information that supplements the error information provided by CPI Communications for a failed verb call.

When XCMGLE is invoked, additional information about the source of the error is returned. If you do not invoke XCMGLE immediately after receiving a parameter check or a product-specific error, the error information is stored until XCMGLE is invoked or another parameter or product-specific error occurs. When XCMGLE is invoked, the error information is cleared. Invoking XCMGLE when no error has occurred or immediately following the last XCMGLE causes zeroes to be returned for all parameters.

One difference between XCMGLE and the other 4690 extensions is that all the parameters included in XCMGLE are used only by the TP to extract information. None of the fields require input from a TP. When the XCMGLE routine is complete, all error information is returned to the TP in the specified parameters issued by XCMGLE.

Note: If the following CPI Communications return codes occur, it is recommended that the TPs issue the XCMGLE routine and store the results:

- CM_parameter check
- CM_product specific error

See the *Systems Application Architecture: Common Programming Interface: Communications Reference* for the values for all CPI Communications return codes.

Format

```
CALL XCMGLE(verb_id,
            CPIC_r_c,
            errcode1,
            errcode2)
```

Parameter
Description
verb_id

Specifies the CPI Communications call that was executed when the last error occurred. The *verb_id* is provided to ensure that the XCMGLE information refers to the correct call. Table 58 contains the *verb_id* values assigned to the calls.

Table 58. *verb_id* Values of CPI Communications Calls

verb_id Value	Pseudonym	Call
1	Accept_Conversation	CMACCP
2	Allocate_Conversation	CMALLC
3	Confirm	CMCFM
4	Confirmed	CMCFMD
5	Deallocate	CMDEAL
6	Extract_Conversation_Type	CMECT
7	Extract_Mode_Name	CMEMN
8	Extract_Partner_LU_Name	CMEPLN
9	Extract_Sync_Level	CMESL
10	Flush	CMFLUS
11	Initialize_Conversation	CMINIT
12	Prepare_To_Receive	CMPTR
13	Receive	CMRCV
14	Request_To_Send	CMRTS
15	Set_Conversation_Type	CMSCT
16	Set_Deallocate_Type	CMSDT
17	Set_Error_Direction	CMSSED
18	Send_Data	CMSSEND
19	Send_Error	CMSERR
20	Set_Fill	CMSF
21	Set_Log_Data	CMSLD
22	Set_Mode_Name	CMSMN
23	Set_Partner_LU_Name	CMSPLN
24	Set_Prepare_To_Receive_Type	CMSPTR
25	Set_Return_Control	CMSRC
26	Set_Receive_Type	CMSRT
27	Set_Sync_Level	CMSSSL
28	Set_Send_Type	CMSST

Table 58. *verb_id* Values of CPI Communications Calls (continued)

verb_id Value	Pseudonym	Call
29	Set_TP_Name	CMSTPN
30	Test_Request_To_Send_Received	CMTRTS

Table 59 contains the *verb_id* values of 4690 extensions.

Table 59. *verb_id* Values of 4690 Extensions

verb_id Value	Pseudonym	4690 Extension
50	Set_Timeout_Value	XCMSTO
52	Enable_Link	XCMELE
53	Conditionally_Disable_Link	XCMCDL
54	Disable_Link	XCMDL
56	Query_Link	XCMQL

CPIC_r_c

Specifies the last CPI Communications return code. The values that follow are a subset of the CPI Communications return codes.

If *CPIC_r_c* contains a zero, no additional error information is available.

CM_PRODUCT_SPECIFIC_ERROR

Indicates that a 4690-specific error occurred on the CPI Communications call.

CM_PROGRAM_PARAMETER_CHECK

Indicates that an error was detected in the CPI Communications parameters.

CM_DEALLOCATE_ABEND

Indicates that the conversation has been terminated because of an error condition. If a TP is written using the Set timeout value function (XCMSTO), the CM_Deallocate_Abend might have occurred because the timeout value expired.

errcode1

Specifies additional 4690 error information that describes the source or type of error that occurred. The information varies according to the CPI Communications return code (*CPIC_r_c*).

CPIC_r_c = CM_PRODUCT_SPECIFIC_ERROR

The 4690 error code is indicated in the *errcode1* field. For information on 4690 errors, see the *4690 OS: Messages Guide*

CPIC_r_c = CM_PROGRAM_PARAMETER_CHECK

When a parameter check occurs, *errcode1* contains one of the following values:

Bad Parameter List 80B21100

Indicates that an incorrect parameter was used on the call. *Errcode2* contains a value that indicates which parameter caused the error.

Parameter Conflict 80B21107

Indicates that the requested call conflicts with the current CPI Communications conversation characteristics.

For example, if you invoke a Confirm (CMCFM) with the *sync_level* set to CM_None, *errcode1* contains a parameter conflict error. A parameter conflict also occurs when you attempt to set the *fill_type* of a CM_Mapped_Conversation to a CM_Fill_Buffer.

CPIC_r_c = CM_DEALLOCATE_ABEND

If the CM_Deallocate_Abend occurred because a call timed out using the Set Timeout Value function (XCMSTO), the 4690 error indicating a call timeout is:

CPIC VERB TIMEOUT 80B21103

If XCMSTO was not used, XCMGLE should not be issued when a CM_Deallocate_Abend return code is received.

errcode2

Specifies additional details describing the type or cause of error. The values vary according to the CPI Communications return code (*CPIC_r_c*).

CPIC_r_c = CM_PRODUCT_SPECIFIC_ERROR

Errcode2 contains a zero.

CPIC_r_c = CM_PROGRAM_PARAMETER_CHECK

Errcode2 contains a value from 1 to 8. The value is determined by the position of the parameter that is not valid in the call.

For example, the conversation ID on all CPI Communications calls is 1 because it is the first field in the parameter list.

In the Receive function call (CMRCV), the *received_length* field position is 5.

A X'99' indicates that the requested call conflicts with the current CPI Communications conversation characteristic. The X'99' is the same as 80B21100 and 80B21107 listed under *errcode1*.

For example, issuing a CMC FM with the *sync_level* set to CM_None causes a X'99' to be stored in the *errcode2* field. If you set the *fill_type* of a CM_Mapped_Conversation to CM_Fill_Buffer, a X'99' occurs.

CPIC_r_c = CM_DEALLOCATE_ABEND

If the deallocate abnormal end occurred because a call timed out using the Set Timeout Value routine (XCMSTO), *errcode2* contains a 0.

Using XCMGLE in 4680 BASIC

Additional error checking is possible for strings used in calls by BASIC TPs. The error checking verifies that the BASIC string is equal to the length that is specified by the parameter. If an error is found, the CM_PROGRAM_PARAMETER_CHECK error is returned as the CPI Communications return code.

Format

```
CALL XCMGLE(verb_id,  
           CPIC_r_c,  
           errcode1,  
           errcode2)
```

Parameter

Description

verb_id

Specifies the call that was running when the last error occurred. The *verb_id* is provided to ensure that the XCMGLE information refers to the correct call.

CPIC_r_c

CM_PROGRAM_PARAMETER_CHECK specifies the error that is returned in the CPI Communications parameters.

errcode1

Is the first parameter containing additional error information. If the value of *errcode1* is -1 to -4, a BASIC error has occurred. The values of *errcode1* and *errcode2* provide further explanation of the CPI Communications return code.

Table 60 on page 224 provided an explanation for error information using BASIC.

errcode2

Indicates which parameter in the command is incorrect by position in the call command. The errors are from 1 to 8, where 1 is the conversation ID for all CPI Communications calls and 8 is the *response_code* field on CMRCV.

Table 60. XCMGLE Error Information Using BASIC

<i>errcode1</i>	Error	Explanation
-1	Not Long Enough	The input string parameter contains fewer characters than the specified length. For example, if you are using CMRCV, and you request that 1000 characters be received into an 80-character field, <i>errcode1</i> is -1.
-2	Lengths Must Match	The specified length of the parameter does not equal the actual length in the input string parameter. The length of the input string parameter is expected to be equal to the length of the specified parameter. For example, if the length of mode name on CMSMN is 6 characters and you supplied a mode name with 5 characters, <i>errcode1</i> is -2.
-3	Too Long	The input string parameter contains more characters than the maximum legal length of the parameter. For example, if you issue an XCMEL function with a link name that contains more than 8 characters, <i>errcode1</i> is -3.
-4	Illegal Null String	The input string parameter does not contain any characters. It is a null string. For example, if you send data using CMSEND from an empty buffer, <i>errcode1</i> is -4.

Query Link Status (XCMQL)

Use the Query Link Status routine (XCMQL) to inquire about the status of a link.

Format

```
CALL XCMQL(link_name,  
           link_name_length,  
           return_code)
```

Parameter

Description

link_name

Specifies the name of the subarea link configuration record, or the peer link configuration record.

link_name_length

Specifies the number of characters in the *link_name*.

return_code

Specifies the library routine return code, as follows:

- 90** Link enabled
- 91** Link disabled
- 92** Link disable pending

Set Timeout Value (XCMSTO)

A TP uses the Set Timeout Value routine (XCMSTO) to set a time limit for the execution of CPI Communications call requests. When the *timeout_value* has been set, CPI Communications calls is timed using that value. If the timer expires before the call completes, the conversation that issued the call is terminated and receives a CM_Deallocate_Abend error.

The *timeout_value* can be set at any time after the conversation has been started. If the *timeout_value* for the conversation is set to zero, then no requests time out for that conversation. The default of the *timeout_value* is zero. If the default is not changed, requests do not time out.

Format

```
CALL XCMSTO(conv_id,  
           timeout_value,  
           return_code)
```

Parameter	Description
-----------	-------------

conv_id	
----------------	--

	Specifies the CPI Communications conversation identifier returned by Initialize_Conversation (CMINIT) or Accept_Conversation (CMACCP).
--	--

timeout_value	
----------------------	--

	Specifies the duration of verb completion time in milliseconds. The <i>timeout_value</i> cannot be greater than a 32-bit integer. If the <i>timeout_value</i> equals 0, calls do not time out.
--	--

return_code	
--------------------	--

	Specifies the CPI Communications return code. The return code field contains these values after completing a call:
--	--

- CM_OK
- CM_PROGRAM_PARAMETER_CHECK The *conv_ID* specifies an unassigned conversation.
- CM_PRODUCT_SPECIFIC_ERROR

Note: XCMSTO does not time out the Allocate_Conversation (CMALLC) call.

Although XCMSTO can time out all CPI Communications calls (except CMALLC), XCMSTO is more beneficial when issuing verbs that must wait for a response from the partner TP. The following verbs require a response:

- CM_RECEIVE with CM_RECEIVE_TYPE set to CM_RECEIVE_AND_WAIT.
- CM_PREPARE_TO_RECEIVE with CM_PREPARE_TO_RECEIVE_type set to CM_CONFIRM or CM_PREPARE_TO_RECEIVE_type set to *sync_level* and *sync_level* set to CM_CONFIRM.
- CM_SEND_DATA with SEND_TYPE set to CM_CONFIRM.
- CM_DEALLOCATE with CM_DEALLOCATE_TYPE set to CM_CONFIRM or CM_DEALLOCATE_TYPE set to *sync_level* and *sync_level* set to CM_CONFIRM.

Starting an LU 6.2 TP

TPs in the store controller can be started in the following ways:

- By an incoming ALLOCATE request from a partner node
- As a primary or secondary application option on the main menu
- From the command line in command mode

From an Incoming ALLOCATE Request

A partner TP from another SNA node can start a 4680 BASIC TP. The partner initiates a conversation by causing an ALLOCATE request to be sent to the operating system. The incoming ALLOCATE contains the name of the TP to be started by the operating system. The names of all TPs that can be started in this way are defined in the Remotely Attachable Local TP Record during communications configuration. If the TP has been defined as remotely attachable, the TP is started as a background application. The TP started in this way should accept the conversation requested by the partner TP. If the operating system does not accept the ALLOCATE request, it ends the conversation.

From the 4690 Main Menu

You can start a TP on the 4690 as a primary or secondary application option on the 4690 main menu. The name of the TP must be defined during controller configuration as the primary or secondary application. A TP that is started from the main menu should initiate the first conversation with a partner TP.

From the Command Line

You can start a TP from the operating system by entering the executable file name on the command line from command mode.

Establishing an SNA Link from LU 6.2 Applications

A TP can establish a link by issuing an Enable Link (XCMELE). You can also establish a link using the COMMUNICATIONS CONTROL panel or the ADXHSNLL.286 application.

After the link is established, the TP can initialize (CMINIT) and allocate (CMALLC) a conversation to communicate with the partner TP. The operating system starts the session as required for the conversation.

For more information on the Enable Link routine, see “Enable Link (XCMELE)” on page 220. See the *4690 OS: User's Guide* for information on the COMMUNICATIONS CONTROL panel.

Ending Links from LU 6.2 Applications

You can disable a link by issuing a Conditional Disable Link (XCMCDL) or a Disable Link (XCMDL) within the TP.

For more information on XCMCDL and XCMDL, refer to “Conditional Disable Link (XCMCDL)” on page 219 and “Disable Link (XCMDL)” on page 219.

To determine the status of a link, use the Query Link Status routine (XCMQL). For more information, refer to “Query Link Status (XCMQL)” on page 224.

Programming Considerations

You can use the following languages on the operating system to call CPI Communications routines and the 4690 extension routines:

- C
- BASIC

Guidance for each of these languages is described later in individual sections. The following guidelines apply to all the languages.

- When a TP issues a CMRCV call, the 4690 implementation of CPI Communications does not return data and conversation status on the same call. Therefore, the transaction program cannot enter the Send-Pending state.
- The mode name and partner LU name that is used on CMSMN and CMSPLN is a record name that must be referenced in the symbolic destination data record during configuration.
- Only the mode name configuration record and partner LU configuration record names are returned in the CMEMN and CMEPLN calls.
- The symbolic destination name that is used in the CMINIT call must be 8 characters. Although it can be padded with blank spaces, 8 characters must be used.
- The operating system accepts a maximum of 512 bytes of CPI Communications log data. Only the first 18 bytes are used for event logging.
- You can write TPs to maintain more than one conversation at the same time. This process is called *concurrent conversations*. For example, a message-routing application can allow a 4690 controller to

send messages between the host and an in-store processor. The 4690 establishes one conversation with the host, and a separate conversation with the in-store processor. The message-routing application sends the messages received from the host conversation to the in-store processor on the concurrent conversation.

The CPI Communications architecture does not limit the number of concurrent conversations that one TP can establish. Although the application is managing concurrent conversations, an application can issue only one CPI Communications verb at a time. Each concurrent conversation must wait when a CPI Communications verb is issued for any of the concurrent conversations. This delay can cause problems when the application issues CPI Communications verbs that can result in long waiting periods.

For example:

```
CMALLC with return_control CM_WHEN_SESSION_ALLOCATED
CMRCV with receive_type CM_RECEIVE_AND_WAIT
CMCFM
```

When you are implementing concurrent conversations, you can avoid delays by using the following alternatives:

```
CMALLC with return_control CM_IMMEDIATE
CMRCV with receive_type CM_RECEIVE_IMMEDIATE
```

Note: If possible, avoid using CMCFM.

C Language

The following notes apply to C programs using CPI Communications routines:

- The pseudonym file CPIC_C.H contains C statements that allow the use of symbolic names (pseudonyms) for various CPI Communications values. This file also contains language-specific calls and is on the 4690 Optionals.
- Before calling a CPI Communications routine, use the *extern* statement to declare the routine as having external linkage, and fully prototype the routine's return value and arguments. CPIC_C.H contains function prototypes for all CPI Communications calls and 4690 extensions.
- When passing an integer value as a parameter, the parameter name should be preceded with an ampersand (&) so that the value is passed by reference.

BASIC

The following notes apply to BASIC programs using CPI Communications routines:

- The pseudonym file CPIC_BAS.DEF contains the BASIC statements that define the BASIC variable types. This file also contains language-specific calls and is on the 4690 Optionals.
- The file CPIC_BAS.EQU contains the BASIC equates that assign a value to each CPI Communications symbolic name (pseudonym).
- The file CPIC_BAS.SUB contains the BASIC function subroutine definitions for the CPI Communications calls.
- CPI Communications integer values are defined as INTEGER*4 data type.
- CPI Communications string variables are defined as STRING data type.

COBOL

The following notes apply to COBOL programs using CPI Communications routines:

- The pseudonym file CPIC-CBL.H contains COBOL statements that allow the use of symbolic names (pseudonyms) for various CPI Communications values. This file also contains language specific calls and is on the 4690 Optionals.
- Because COBOL does not support the underscore character (), the underscores in COBOL pseudonyms are replaced with dashes (-). For example, COBOL programmers would use CM-IMMEDIATE as a pseudonym value name in their programs instead of CM_IMMEDIATE.

- Use the */litlink* option when compiling COBOL TPs to correct CPI Communications calls.

Chapter 12. Designing an X.25 Application

This chapter assumes that you are familiar with X.25 concepts and protocols. For more information on X.25 see the publications available on the Toshiba support site.

The operating system provides an X.25 API. You can use this API to write programs that use the X.25 protocol to communicate directly over an X.25 network. For example, you can write an application that communicates with another application located at a node on an X.25 network. (Previously, the 4680 Operating System supported X.25/SNA communications using the Qualified Logical Link Control (QLLC) protocol over an X.25 virtual circuit.)

In addition to this support, the communications menu of the 4690 store controller offers an X.25 API selection. Use this selection to give your X.25 application programs direct access to an X.25 network. For information about installing and configuring the X.25 API, see the *4690 OS: Planning, Installation, and Configuration Guide*.

Hardware Requirements for the X.25 API

X.25 support for both the X.25 API and X.25/SNA requires the X.25 Interface adapter. This adapter belongs to the ARTIC family.

The adapter is a single-slot, full-length adapter. It contains a user memory of 512 or 1024 KB (DRAM) and 16 KB of programmable read-only memory (PROM). Before the adapter can be used, the realtime control microcode (RCM), which is stored in ADX_SPGM as ICAAIM.COM during hardware installation, must be loaded on the card during IPL. A maximum of two X.25 interface cards may be configured per controller.

Three network interface cables are available for the adapter:

- X.21 bis/V.24 (1 port) asynchronous/synchronous 19.2 Kbps (electrically compatible with RS-232-C)
- X.21 (1 port) synchronous interface 64 Kbps (electrically compatible with RS-422-A)
- X.21 bis/V.35

Note: The 4690 store controller supports only the X.21 bis/V.24 cable option.

Software Requirements for the X.25 API

Support for X.25 by the X.25 Interface adapter requires Realtime Co-Processor Extended Services (5688-044).

Understanding the X.25 API

The X.25 API enables applications to exchange information over an X.25 network using the network layer services of the operating system, as illustrated in Figure 9 on page 230. These services comply with the International Telegraph and Telephone Consultative Committee (CCITT) 1980/1984/1988 recommendations *ISO 7776 DTE Data Link Procedures for X.25* and *ISO 8208 Packet Layer Protocol for DTE*

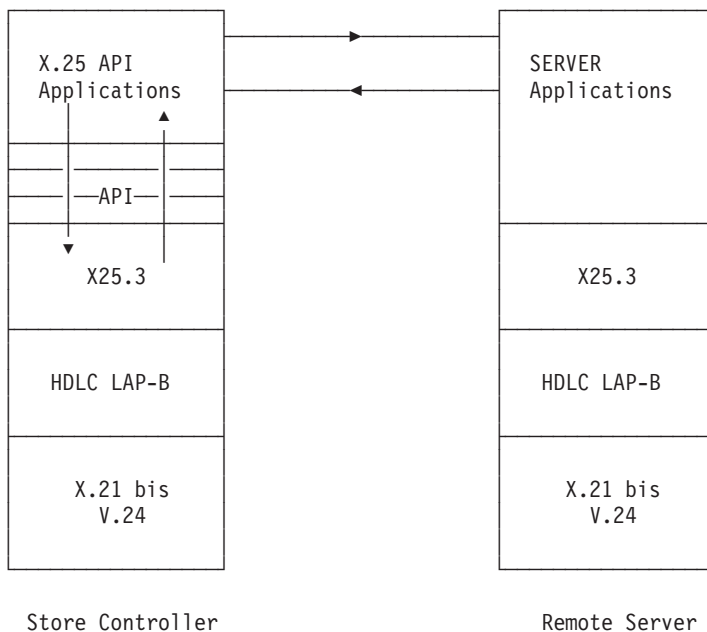


Figure 9. API Using Network Layer Services

Hardware Support for the X.25 API

The store controller supports one or two X.25 Interface adapters. Each adapter provides 15 virtual circuits, which can be any combination of permanent virtual circuits (PVC) and switched virtual circuits (SVC). These 15 virtual circuits can be shared with SNA X.25. However, SNA X.25 is limited to four virtual circuits per adapter. An application can have a combination of virtual circuits open that does not exceed these limits. The API must know the number of adapters and permanent or switched virtual circuits. This information is defined in the controller configuration. See the *4690 OS: Planning, Installation, and Configuration Guide* for information on the configuration of the controller for the X.25 API.

Software Support for the X.25 API

The API is compatible with the multitasking features provided by the operating system. That is, several applications running concurrently can use the API services independently. The X.25 application programs that you write gain access to X.25 network services by calling the X.25 API verbs. The following verbs are available (see Chapter 13, "X.25 API Verb Reference" for a detailed description of the verbs and their parameters).

XLOAD	Requests the API to load the communications drivers and listen for incoming calls
XOPEN	Requests the API to format and send a Call Request packet to establish an SVC
XSEND	Requests the API to format and send as many data packets as necessary on a previously established virtual circuit
XRECEIVE	Requests the API to deliver the received data or control information
XCLOSE	Requests the API to format and send a Clear Request packet to disconnect an SVC
XEVENT	Requests the API to deliver information from a received Clear, Interrupt, or Reset Indication packet
XOPENREC	Requests the API to deliver information from an Incoming Call packet

XOPENACC	Requests the API to format and send a Call Accepted packet (in response to an Incoming Call packet)
XIT	Requests the API to format and send an Interrupt packet
XRESET	Requests the API to format and send a Reset Request packet
XALLOC	Requests the API to allocate a PVC
XDEALLOC	Requests the API to deallocate a PVC
XSTO	Sets a timeout value for verbs
XWAIT	Used with BASIC only, multiple event wait. (Programs written in COBOL or C achieve the same result by calling the ADX_CWAIT function.)

For a given application, the execution of each requested API verb is synchronous. However, the ADX_CWAIT operating system Interface call allows X.25 API programs to wait on incoming X.25 data simultaneously with other asynchronous events. (In the case of programs written in 4680 BASIC, this *XWAIT* verb performs this function.) In particular, the call enables an application (for example, an electronic funds transfer (EFT) controller application) to receive messages from both the X.25 link and from other applications through a pipe. See “ADX_CWAIT (C and COBOL Only, Multiple Event Wait)” on page 252 for more information.

When the API receives a Clear, Interrupt or Reset Indication packet from the network, it formats and immediately sends a corresponding confirmation packet, independently of any application program request.

Implementation of X.25 Protocol Characteristics

The X.25 protocol characteristics confirm delivery of data packets, mark the data packet as belonging to the same application as a previous data packet, and qualify the data in the packet. These characteristics are :

- Management of the Delivery Confirmation indicator (D-bit):
A data packet with the D-bit set to 1 requires a delivery confirmation from the receiving data terminal equipment (DTE), while a data packet with the D-bit set to 0 needs only a confirmation from the network access point. The API supports both sending and receiving the D-bit (see “XSEND” on page 239 and “XRECEIVE” on page 241).
- Management of the More Data mark (M-bit):
A data packet with the M-bit set to 1 means that the next packet belongs to the same application message (complete logical sequence of data). The API provides M-bit management (see *XSEND* and *XRECEIVE*).
- Management of the Qualifier bit (Q-bit):
A data packet with the Q-bit set to 1 qualifies the data in the packet. The Q-bit set indicates that the data is significant for a device attached to the remote DTE.

Principal Features of the X.25 API

The X.25 API enables store controller applications to share the X.25 subsystem with existing X.25/SNA links. Figure 10 on page 232 shows existing components of the operating system. It also shows the components providing X.25 API support.

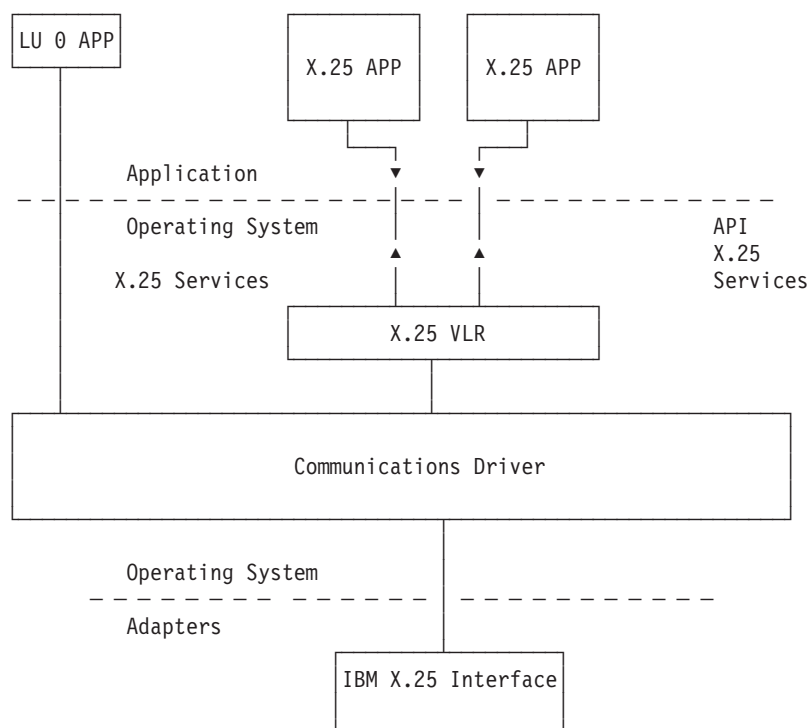


Figure 10. Addition of X.25 API Applications

Summary of Features

The following summarizes the features of the X.25 API:

- The X.25 API allows user applications written in C, 4680 BASIC, or COBOL, direct access to X.25 services. In addition, it enables you to use applications written in other languages that conform to the calling conventions of C, 4680 BASIC, or COBOL (for example, assembly language).
- An X.25 API application and an X.25 SNA application can both access an X.25 network through the same X.25 Interface adapter.
- All services that are required to implement an open systems interconnection (OSI) transport layer are supported by the API. This support allows the implementation of an application to provide the upper layers of the OSI model. However, no actual transport implementation is provided.
- An X.25 application can be loaded dynamically on receipt of a Call request from a remote application using the user data contained within the request. See “Remote Loading of Applications” on page 258 and “XOPEN” on page 236 for more information.
- If not already loaded, the communications driver code is loaded and started on the first application XLOAD request. Note that the XLOAD verb can be executed by a “start of day” application to enable X.25 API. See “XLOAD” on page 236 for more information.
- The maximum size of data that can be processed at one time by the API is 32 KB. To send more, an X.25 application must issue a chain of sends, setting the More Data mark (M-bit) on all but the last XSEND.
- To minimize the problem of “call races” during the establishment of SVCs, the API allocates the highest free channel (from its permitted range) to outgoing calls, and the data circuit-terminating equipment (DCE) allocates the lowest free channel to incoming calls. If only one free channel is available, the incoming call has priority and the outgoing call is rejected with a “lack of circuit resources” error.

- The previous limits for X.25 SNA of four virtual circuits per adapter are extended to allow each adapter to support up to 15 virtual circuits at any one time. The circuits are shared between X.25/SNA and X.25 API. X.25/SNA can still use only four of the circuits.

If 15 channels or virtual circuits are configured for use by X.25 API (that is, none are configured for X.25/SNA), up to 15 X.25 applications, each accessing a single virtual circuit, can run concurrently.

The maximum number of virtual circuits available to a single application is the number of virtual circuits configured, minus any in use at that time.

Error Logging and Recovery

User errors detected by the API for incorrect X.25 function calls are not logged, but appropriate error codes are returned to the requesting application. These return codes are listed in Appendix D, “X.25 API Reference Information.” All error codes are returned in the *return_code field*. Non-API errors are logged. You should follow normal operating system problem determination and resolution procedures for non-API errors.

System Log Scan

They are logged in the SNA communications driver bucket.

Some Common Reasons for Problems

The system log function formats errors that are logged by the X.25 API function.

The following lists some common causes for problems you may have with the X.25 API.

- Mismatch of channel numbers between 4690 and network (line record)
- Wrong DTE address used on call
- Call request contained remote application name by mistake
- Network not configured correctly for DTE address
- Network not configured correctly for end-to-end PVC connection
- Communications driver not loaded (*XLOAD* verb not executed)

Items to Check

The following list provides items to check when you have problems with X.25 API function. Use the Event Type, Cause Code, and Diagnostic Code returned by the *XEVENT* verb to determine the cause of the problem. The Event Type, Cause Codes, and Diagnostic Codes are documented in Appendix D, “X.25 API Reference Information,” on page 347.

1. Turn on the line trace on the 4690 and check if Receive Ready (RR) polls are being exchanged.
2. Outgoing call:
 - a. Check that the call request is being transmitted (4690 line trace).
 - b. Check DTE addresses and the rest of call data.
3. Incoming call:
 - a. Check that the call was received on a configured channel.
 - b. Check that the called DTE address does not clash with X.25/SNA.
 - c. Check if the first byte of call user data (match byte) is X'00'
 - d. Check if the second byte of call user data (match type) is X'00', or else that the application is being loaded.
 - e. Check the application's call acceptance criteria.
4. Check the PVC line trace to see if a Reset request was acknowledged.

X.25 Communications Line Trace

The 4690 Trace facility provides X.25 API line traces. The trace facility includes an option for capturing a communications line trace. See the *4690 OS: Messages Guide* for more information on the Trace facility.

Chapter 13. X.25 API Verb Reference

The X.25 API defines a set of X.25 API verb library routines (VLRs) that provide a synchronous interface to X.25 services. Applications can request the initiation of a PVC or a SVC between themselves and remote X.25 applications. The term *X.25 API* is used in this chapter to refer to functions or services performed by X.25 services, the communication driver, or both.

VLRs are available in these libraries:

- ADXHS51L.L86 for 4680 BASIC
- ADXHS52L.L86 for COBOL
- ADXHS53L.L86 for C

Procedure definitions and constant definitions are available in the following include files:

For 4680 BASIC:

- ADXHS5KF.L86 procedural interfaces
- ADXHS5LF.J86 declaration of return code variables
- ADXHS5MF.J86 initialization of return code variables

For C:

- ADXHS5LC.L86 return code and other constants
- ADXHS5KX.X86 procedural interfaces

For COBOL:

- X25N_CBL.H return codes

An application gains access to X.25 services by initializing the communications (XLOAD) and then establishing an SVC or allocating a PVC. Each circuit is referenced by the name of its virtual circuit record (VCR). This reference is a configuration record that specifies the line name, the type of circuit, and the called address. Many of the supported verbs have optional parameters that you can use to override the settings associated with the VCR configuration.

CALL Syntax for X.25 API Verbs

In the description of each API verb, the general syntax format shown in Figure 11 is used.

```
CALL VERB  (parm0,  
            parm1,  
            parm2,  
            .  
            .  
            .  
            parmN)
```

Figure 11. General Syntax of API Verbs

parm0, *parm1*, *parm2*, ..., *parmN* are the parameters required for the particular callable verb. There are no optional parameters in the API callable verbs.

The parameters are described as supplied parameters or returned parameters and are passed by reference.

- Supplied parameters contain arguments whose values are supplied by the program when it issues the verb.
- Returned parameters contain arguments whose values are returned to the program when it resumes processing.

- Sometimes a parameter is both supplied and returned. See Appendix D, “X.25 API Reference Information” for an explanation of these cases.

Supported Verbs

The following API verbs are available to an X.25 application. A short description of each verb is followed by its syntax. The description of each verb is relevant to all supported languages. The syntax applies to C, 4680 BASIC, and COBOL languages.

XLOAD

An application must call the XLOAD verb for each X.25 line to initialize X.25 communications over that line. If two X.25 adapters are being used, the application must issue this verb twice, once for each adapter or line. The X.25 API ensures that the necessary communications driver is loaded and initialized. Issuing this verb is not necessary if the drivers are already installed.

An application may not use any other X.25 API verb until the necessary communications driver has been loaded. Loading can be performed by a background controller task issuing an XLOAD verb. All applications should issue the XLOAD verb to verify that the communications driver is loaded before attempting to use it. Once the drivers are loaded they stay resident in memory even when not active. To free up the memory they use, you can enable and then disable a dummy QLLC link.

Note: The communications driver cannot be unloaded using the X.25 API.

Format

```
CALL XLOAD      (line_name,  
                 line_name_length,  
                 return_code)
```

Parameters

***line_name* (supplied)**

Specifies the name of the X.25 line (8 alphanumeric characters), as defined by the Line Name in configuration. This parameter is mandatory.

***line_name_length* (supplied)**

Specifies the length of the line name of the X.25 line. This parameter is mandatory.

***return_code* (returned)**

Completion return code set by the API and indicating the result of the verb execution. See “X.25 Return Code and Verb Cross-Reference Table” on page 259 for a complete list of return codes.

XOPEN

The application issues an XOPEN call to request the network layer to establish an SVC between the local DTE (the store controller) and a called DTE (for example, a card authorization server). The XOPEN verb establishes an outgoing call on an SVC. A mandatory parameter for this verb is the name of a configured VCR. Various parameters, such as *called_address*, can be passed that override the settings associated with the configured VCR.

During the XOPEN call, the calling X.25 API application can request delivery confirmation services. This request allows the application to send data with delivery confirmation from the remote DTE. If this service is requested, the application can later use XSEND to set the delivery confirmation indicator. It is the responsibility of the X.25 API to hold the XSEND call until the remote DTE confirms delivery. The XSEND verb completes successfully when the remote DTE confirms delivery. If delivery is not confirmed, the XSEND verb fails.

If the remote application requests delivery confirmation, the X.25 API is responsible for that confirmation. The X.25 application does not itself handle the delivery confirmation request. The facilities buffer provided for delivery confirmation must be at least 109 bytes. The call user data buffer must be at least 128 bytes for a “fast select” call or at least 16 bytes otherwise.

The XOPEN verb can also request that an application be loaded at the called 4690 store controller. See “Remote Loading of Applications” on page 258 for more details.

Format

```
CALL XOPEN    (vcr_name,
               vcr_name_length,
               remote_application,
               remote_application_length,
               D_bit_services,
               called_address_length,
               called_address,
               facilities_field_length,
               facilities_field,
               call_user_data_length,
               call_user_data,
               connection_ID,
               return_code)
```

Parameters

vcr_name (supplied)

Specifies the name of the X.25 VCR (up to 8 alphanumeric characters), as defined in configuration. This parameter is mandatory. Additional XOPEN parameters can be associated with the VCR name at configuration time. See the *4690 OS: Planning, Installation, and Configuration Guide* for information about configuration.

vcr_name_length (supplied)

Specifies the length of the VCR_name in bytes. This parameter is mandatory.

remote_application (supplied)

Specifies the logical name of the remote 4690 application that is to be loaded. The loaded 4690 application must receive the call by issuing the XOPENREC verb. The remote application name in the VCR, if present, is used if this parameter is null.

remote_application_length (supplied)

Specifies the length of the remote application, which is the logical name of the remote 4690 application.

D_bit_services (supplied/returned)

- Supplied

Specifies that D_bit (delivery confirmation) services can be used during the call.

0 = Delivery confirmation services are requested.

1 = Delivery confirmation services are not requested.
- Returned

Provides the status of the D_bit returned by the called application (XOPENACC).

0 = D_bit services cannot be used during the call.

1 = D_bit services can be used during the call.

called_address_length (supplied)

Specifies the length (from 0 to 15 bytes) of the *called_address*, which contains the Packet Switching Data Network (PSDN) address of the called DTE.

XOPEN

called_address (supplied)

Specifies the PSDN address of the called DTE. Ignored if the *called_address_length* is 0. Otherwise, overrides the called address defined in the VCR.

facilities_field_length (supplied/returned)

- Supplied
Specifies the length (from 0 to 63 bytes CCITT 1980 or from 0 to 109 bytes CCITT 1984/1988) of the X.25 *facilities_field* that the network layer inserts in the Call Request packet.
- Returned
Specifies the length (from 0 to 109 bytes) of the *facilities_field* in the packet received by the network layer in response to the Call Request packet.

Note: A DTE does not receive Call Connected or Clear Indication packets containing an X.25 *facilities_field*, if the PSDN contract does not specify compliance with the CCITT 1984 recommendation.

facilities_field (supplied/returned)

Note: The *facilities_field* must be capable of holding 109 bytes. Because supplied and returned facilities information is passed in this field, it must be large enough to hold the maximum size *facilities_field*.

- Supplied
Specifies the X.25 *facilities_field* to be inserted in the Call Request packet. Must be formatted in compliance with the CCITT recommendation.
Examples of X.25 facilities:
 - “Fast select” with or without restriction
 - Closed user groupIgnored if the *facilities_field_length* is 0; otherwise, overrides the *facilities_field* associated with the VCR name.
- Returned
Specifies the *facilities_field* received by the network layer in the Call Connected or Clear Indication packet. The area specified must be large enough to receive the maximum size *facilities_field* that can be received (for example, 63 bytes for CCITT 1980 and 109 bytes for CCITT 1984/88).

call_user_data_length (supplied/returned)

- Supplied
Specifies the length (in bytes) of *call_user_data*. The network layer uses this value to format the Call Request packet. Valid values are from 0 to 16 when the “fast select” service is not requested, and up to 128 for a “fast select” call.
- Returned
Specifies the length (from 0 to 128 bytes) of the user data in the packet received by the network layer in response to the Call Request packet (Call Connected or Clear Indication packet).

call_user_data (supplied/returned)

Note: The *call_user_data* field must be capable of holding 128 bytes. Because supplied and returned *call_user_data* information is passed in this field, it must be large enough to hold the maximum size *call_user_data*.

- Supplied
Specifies the user data. Ignored if the *call_user_data_length* is 0; otherwise, overrides the call user data associated with the VCR name. If the remote DTE is a 4690 store controller, and a remote application name is provided in this call, this field identifies the application that is to be loaded at the remote 4690 store controller. In this case, bytes 1 to 8 carry the application name

and are reserved for use by the X.25 API. You can use byte 0 and bytes 9 to 15. The Call Connected response from the remote 4690 store controller is delayed until the remote application is loaded and issues the XOPENREC and XOPENACC verbs.

The following rules determine the contents of bytes 1 to 8 of the user data field. The highest priority is 1.

Table 61. User Data Field Priority

Priority	Contents of bytes 1 to 8
1	Remote application name from XOPEN verb, if present.
2	Remote application name from VCR, if present.
3	Contents of bytes 1 to 8 of XOPEN <i>call_user_data</i> field, if present. This could be a remote application name.
4	Contents of bytes 1 to 8 of the <i>call_user_data</i> field in the VCR, if present.

- Returned

Specifies the user data received by the network layer in the Call Connected or Clear Indication packet.

Note: The first byte of the call user data can be a protocol identifier (for example, Call Request packet sent by a packet assembler/disassembler (PAD)).

connection_ID (returned)

Identifies the SVC, if the verb is executed successfully (return code = OK), and a Call Accepted packet was received at the network level. Data packets can now be sent or received on the SVC.

return_code (returned)

Completion return code set by the X.25 API and indicating the result of the verb execution. See Appendix D, “X.25 API Reference Information” for a complete list of the possible return codes.

XSEND

The X.25 application issues an XSEND call to send data on a virtual circuit established or allocated for it. The data sent by the XSEND verb can cause many 128-byte data packets to be sent by the X.25 API (the maximum packet data size is 128 bytes). The API sets the More Data bit (M-bit) in all but the last packet sent. If the X.25 application chooses to send its own sequence of data records, it sets the *M_bit_indicator* on all but the last record. All but the last record must be 128 bytes long.

The maximum amount of data that may be sent on one XSEND verb is 32 767 bytes. To send more than 32 767 bytes, the application issues as many chained XSEND calls as necessary. Chained XSEND calls need not be consecutive; the application may issue other verb calls within a chained XSEND call sequence. The application can chain the calls logically by setting the M_bit indicator.

If the amount of data specified in a call is less than the packet size, the X.25 API formats a data packet with the M-bit set to the value of the *M_bit_indicator* parameter. Otherwise, the X.25 API formats and sends a sequence of data packets. All the packets of the sequence, except the last one, have the M-bit set to 1. The last packet has the M-bit set to the value of the *M_bit_indicator* parameter (1, if not the last chained XSEND call in a sequence; otherwise, 0).

The X.25 application can set the delivery confirmation indicator (D-bit) if this service was requested on the XOPEN or XOPENREC verb. Delivery confirmation requires the remote DTE to confirm delivery of the data. The D-bit need be set only in the last data packet of a sequence that is sent. Control returns to the X.25 application when:

- Data is accepted by X.25 API (No D-bit used).
- The other end acknowledges the packet with the D-bit set (D-bit used).

XSEND

The Qualifier bit (Q-bit) is user-settable and has significance from end-to-end only. The More bit (M-bit) is similar.

Format

```
CALL XSEND    (connection_ID,  
               data_length,  
               data,  
               Q_bit_indicator,  
               M_bit_indicator,  
               D_bit_indicator,  
               data_bytes_sent,  
               return_code)
```

Parameters

connection_ID (supplied)

Specifies the identifier of the virtual circuit on which data is to be sent. This *connection_ID* must have been returned on a previous XOPEN, XOPENREC, or XALLOC call.

data_length (supplied)

Specifies the length (from 0 to 32 767 bytes) of the data to be sent.

data (supplied)

Specifies the data to be sent.

Q_bit_indicator (supplied)

Specifies the Q-bit setting in the data packets to be assembled (if the Q-bit is set to 1, the data is control data). This data is significant only to the sending and receiving DTEs (for example, control packets between a Packet-DTE and a PAD).

M_bit_indicator (supplied)

Specifies whether the last packet to be assembled must have the M-bit set to 1 or 0. (If set to 1, the X.25 application must be designed in such a way that the XSEND verb is called at least one more time with the *M_bit_indicator* parameter set to 0). Valid values are 0 or 1.

D_bit_indicator (supplied)

A data packet with the D-bit set to 1 requires a delivery confirmation from the receiving DTE, while a data packet with the D-bit set to 0 needs only a confirmation from the X.25 API.

data_bytes_sent (returned)

Reports back the number of bytes of data sent by the completed verb call.

This number can be used in the case where an interrupt is received in the middle of sending data. The XSEND verb completes with an XA.EVENT return code, and this field is set to the number of bytes that were actually sent by the X.25 API. After issuing an XEVEN verb to receive the interrupt data, the X.25 application can issue the XSEND verb again. However, the data pointer must first be incremented by the *data_bytes_sent* value returned in the previous XSEND.

return_code (returned)

Completion return code set by the X.25 API indicating the result of the verb call. For a complete list of the possible return codes, see Appendix D, "X.25 API Reference Information."

Note: The XA.OK return code indicates that the X.25 API has accepted your request. The X.25 API does not guarantee that the data was delivered to the X.25 network on the remote DTE. The X.25 application program performs this function.

XRECEIVE

The XRECEIVE verb is used to receive information sent by a remote DTE. Because the XRECEIVE verb is synchronous, an X.25 application can specify the delay the X.25 API observes before returning (if no data or control packets were received). The packets received can be:

- Data packets
- Clear Indication packet
- Interrupt packet
- Reset Indication packet

Receiving a Clear or Reset packet clears any received data not copied into the receive buffer. Any data in the receive buffer cannot be guaranteed. Receiving an Interrupt packet does not affect received data. The X.25 application is not aware of any underlying delivery confirmation requested by the remote DTE or performed by the API.

Format

```
CALL XRECEIVE (connection_ID,
               data_buffer_size,
               call_receipt_delay,
               data_length,
               data,
               Q_bit_indicator,
               M_bit_indicator,
               return_code)
```

Parameters

connection_ID (supplied)

Specifies the identifier of the virtual circuit on which data or control packets can be received. It must have been returned on a previous XOPEN, XOPENREC, or XALLOC call.

data_buffer_size (supplied)

Specifies the length (in bytes) of the application-supplied buffer (that is, the amount of data that the X.25 application is able to accept).

receipt_delay (supplied)

Specifies the delay (in seconds) that the X.25 API observes before returning, if no data or control packets have been received. The return code is NOTHING_RECEIVED_IN_DELAY (see Appendix D, "X.25 API Reference Information" for the value). Valid values for the delay are from -1 to 32 767, where -1 means wait indefinitely.

data_length (returned)

Specifies the length (in bytes) of the data returned to the X.25 application. Can be smaller than or equal to the data buffer size.

data (supplied)

Specifies the buffer that the X.25 API places the received data into.

Q_bit_indicator (returned)

Specifies whether the received data is control data (each packet of the returned sequence has its Q-bit set to the same value).

M_bit_indicator (returned)

Specifies whether the application should issue at least one more XRECEIVE call (set to 1 whenever the supplied buffer is too short for a complete packet sequence, or an incomplete packet sequence is received within the specified delay).

return_code (returned)

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, "X.25 API Reference Information."

XRECEIVE

Note: When a data packet with the D-bit set to 1 is received at the X.25 API level, the X.25 API confirms the delivery of the packet to the sending DTE (see Figure 12). Delivery is confirmed when the application data is copied into the application receive buffer.

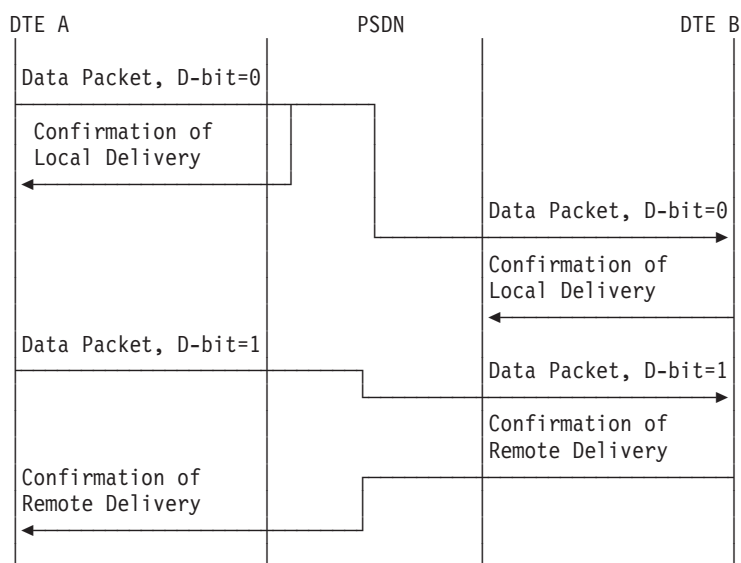


Figure 12. Confirmation of Local/Remote Delivery of a Data Packet

XCLOSE

The application issues an XCLOSE call to request the network layer either to disconnect (clear) an SVC that was previously established or to reject an incoming call. XCLOSE normally sends a Clear packet on the circuit, except when the circuit is already cleared. If a Clear is sent but a timeout occurs before a Clear Confirm is received, XCLOSE returns a successful return code.

After an XOPEN or XOPENREC is issued, XCLOSE must be issued to free the virtual circuit for reuse. The X.25 API does not provide any facility for the unloading of the communications driver once it is loaded.

Format

```
CALL XCLOSE (connection_ID,
             facilities_field_length,
             facilities_field,
             clear_user_data_length,
             clear_user_data,
             cause_code,
             diagnostic_code,
             return_code)
```

Parameters

connection_ID (supplied)

Specifies the identifier of the SVC to be cleared. This ID must have been returned on a previous XOPEN or XOPENREC call.

facilities_field_length (supplied/returned)

- Supplied

Specifies the length (from 0 to 63 or 109 bytes) of the X.25 *facilities_field* that the network layer will insert in the Clear Request packet to be formatted and sent.

- Returned

Specifies the length (from 0 to 109 bytes) of the X.25 *facilities_field* contained in the Clear Confirmation packet.

facilities_field (supplied/returned)

- Supplied

Note: The *facilities_field* must be capable of holding 109 bytes. Because supplied and returned facilities information is passed in this field, it must be large enough to hold the maximum size *facilities_field*.

Specifies the X.25 *facilities_field* (in compliance with the X.25 CCITT recommendation, for example, “fast select”) that the network layer inserts in the Clear Request packet to be formatted and sent. Ignored if the *facilities_field_length* is 0.

- Returned

Specifies the X.25 *facilities_field* (in compliance with the X.25 CCITT recommendation, for example, “fast select”) contained in the Clear Confirmation packet.

The area specified must be large enough to receive the maximum size *facilities_field* that may be received (for example, 63 bytes for CCITT 1980 and 109 for CCITT 1984/88).

Note: A DTE will receive no Clear Confirmation packet containing an X.25 *facilities_field* if the PSDN contract does not specify compliance with the CCITT 1984 recommendation.

clear_user_data_length (supplied)

Specifies the length (from 0 to 128 bytes) of the user data to be inserted by the network layer in the Clear Request packet.

Note: If the clear user data length is different from 0, the “fast select” facility must be requested.

clear_user_data (supplied)

Specifies the user data that the network layer inserts in the Clear Request packet to be formatted and sent. The “fast select” facility is required. Ignored if the *clear_user_data_length* is 0.

cause_code (supplied)

Specifies the value of the cause byte to be inserted in the Clear Request packet. The cause byte set by the application must be either equal to 0, or greater than or equal to 128 (bit 8 set to 1).

diagnostic_code (supplied)

Specifies the value of the diagnostic byte to be inserted in the Clear Request packet.

return_code (returned)

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, “X.25 API Reference Information.”

XEVENT

The XEVENT verb gives access to the information in the Clear, Reset, and Interrupt packets. Use XEVENT to investigate an XA.EVENT return code or anytime XOPEN or XOPENREC fails. If the XOPEN or XOPENREC verb fails following an established network connection (for example, after a Clear Indication packet from the remote DTE), then a valid connection ID is returned from the XOPEN or XOPENREC verb, even though the return code indicates a failure. The connection ID returned from a failed XOPEN or XOPENREC cannot be used with any other verb. If another verb is used, an *INVALID_CONNECTION_ID* is returned. In any case, the XEVENT verb requires an established connection ID.

XEVENT

If the API receives a Reset Indication or Clear Indication, any outstanding receive data is cleared. Any data or facilities associated with the Reset, Interrupt, or Clear are retained by the API and returned on a subsequent XEVENT verb.

Only a single exception event is saved for return to the application. Events are overridden in priority. That is, Clear can overwrite Reset, and Reset can overwrite Interrupt.

Format

```
CALL XEVENT    (connection_ID,
                event_type,
                facilities_field_length,
                facilities_field,
                data_length,
                data,
                cause_code,
                diagnostic_code,
                return_code)
```

Parameters

connection_ID (supplied)

Specifies the identifier of the virtual circuit over which the event is received. This connection ID must have been returned on a previous XOPEN, XOPENREC, or XALLOC call.

event_type (returned)

Specifies the event that occurred:

- CLEAR_INDICATION (value 1)
- RESET_INDICATION (value 2)
- INTERRUPT_INDICATION (value 3)

facilities_field_length (returned)

Specifies the length (from 0 to 109 bytes) of the X.25 *facilities_field* contained in a Clear Indication packet received by the network layer.

facilities_field (returned)

Note: The *facilities_field* must be capable of holding 109 bytes. Because returned facilities information is passed in this field, it must be large enough to hold the maximum size *facilities_field*.

Specifies the X.25 *facilities_field* (in compliance with the X.25 CCITT recommendation, for example, "fast select") contained in a received Clear Indication packet. Ignored if the *facilities_field_length* is 0.

Note: A DTE will receive no Clear Indication packet containing an X.25 *facilities_field* if the PSDN contract does not specify compliance with the CCITT 1984 recommendation.

data_length (returned)

Specifies the length (in bytes) of the data contained in the received Clear Indication or Interrupt packet. Valid values are from 0 to 128 for a received Clear Indication, and from 1 to 32 for a received Interrupt.

data (returned)

Specifies the data contained in the Clear Indication or Interrupt packet. Ignored if the *data_length* is 0.

cause_code (returned)

Specifies the value of the cause byte contained in the received Clear or Reset Indication packet.

***diagnostic_code* (returned)**

Specifies the value of the diagnostic byte contained in the received Clear or Reset Indication packet.

***return_code* (returned)**

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, “X.25 API Reference Information.”

XOPENREC

The X.25 application program issues an XOPENREC call to receive an incoming call. If an incoming call is received and there are no XOPENREC verbs queued, the call is cleared unless the call specifies an application name to be loaded. See “Remote Loading of Applications” on page 258 and the XOPEN verb for more details. The X.25 application can choose to receive any call or a particular call by specifying acceptance criteria that match data in the incoming call request. These criteria are as follows:

- The first byte of the user data contained in the incoming call packet
- The *calling_address* field of the incoming call packet
- The *called_address* field of the incoming call packet

These fields can be specified as acceptance criteria parameters to the XOPENREC verb. Each has an associated parameter that tells the X.25 API whether to match that criterion and the length of the criterion to use. If *first_byte_of_user_data*, *calling_address_length*, or *called_address_length* are zero then the related criterion is not used. If none of the criteria are specified, the X.25 API treats the XOPENREC as a “wildcard” request. It passes the request to the first incoming call request that does not have another XOPENREC with matching acceptance criteria waiting for it.

If the application chooses to accept the incoming call, it uses the XOPENACC verb. If the application chooses to reject the incoming call, it uses the XCLOSE verb. If the remote DTE requested delivery confirmation services, the application is informed through a returned parameter on XOPENREC.

Format

```
CALL XOPENREC (first_byte_of_user_data,
               first_byte_of_user_data_value,
               D_bit_services,
               calling_address_length,
               calling_address,
               called_address_length,
               called_address,
               call_receipt_delay,
               facilities_field_length,
               facilities_field,
               call_user_data_length,
               call_user_data,
               connection_ID,
               return_code)
```

Parameters

***first_byte_of_user_data* (supplied)**

Specifies whether the first byte of the user data contained in an incoming call packet must match the value supplied in the *first_byte_of_user_data_value* parameter. Valid values are 0 or 1.

***first_byte_of_user_data_value* (supplied)**

Specifies the value that the X.25 API must compare with the first byte of the user data contained in an incoming call packet. Ignored if the *first_byte_of_user_data* parameter is set to 0.

XOPENREC

D_bit_services (returned)

Tells the called application whether the calling application is requesting *D_bit* services. The called application either confirms or denies the request using the *D_bit_services* parameters of the XOPENACC verb.

calling_address_length (supplied)

Specifies the length (from 0 to 15 bytes) of the application-supplied calling address.

calling_address (supplied)

Specifies the application-supplied calling address to be compared with the calling address in an incoming call packet. Ignored if the *calling_address_length* is 0.

called_address_length (supplied)

Specifies the length (from 0 to 15 bytes) of the application-supplied called address.

called_address (supplied)

Specifies the application-supplied called address to be compared with the called address in an incoming call packet. Consists of the principal and the complementary addresses. Ignored if the *called_address_length* is 0.

call_receipt_delay (supplied)

Specifies the delay (in seconds) that the API observes before returning control to the application when no valid incoming call packet is received. Valid values are from -1 to 32 767, where -1 means to wait indefinitely.

facilities_field_length (returned)

Specifies the length (from 0 to 109 bytes) of the *facilities_field* that is contained in the incoming call packet received by the network layer.

Note: A DTE will receive no incoming call packet containing an X.25 *facilities_field* if the PSDN contract does not specify compliance with the CCITT 1984 recommendation.

facilities_field (returned)

Specifies the *facilities_field*, if any, as received in the incoming call packet by the network layer.

Note: This field must be large enough to hold the maximum size *facilities_field* that can be returned.

call_user_data_length (returned)

Specifies the length (from 0 to 128 bytes) of the *call_user_data* contained in the incoming Call packet received by the network layer.

.call_user_data (returned)

Note: The *call_user_data* field must be capable of holding 128 bytes. Because returned call user data information is passed in this field, it must be large enough to hold the maximum size call user data.

Specifies the user data received by the network layer in the incoming call packet.

connection_ID (returned)

Identifies the SVC, if the verb completes successfully (return code = OK).

return_code (returned)

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, "X.25 API Reference Information."

XOPENACC

The XOPENACC verb is called to accept the incoming call request following a successful XOPENREC verb call. The called X.25 API application can indicate that it intends to send data that requires delivery

confirmation from the calling DTE. This service can be requested on the XOPENACC verb. See XOPEN for more details on delivery confirmation. To reject an incoming call, the application would issue an XCLOSE. See “XCLOSE” on page 242 for details.

Format

```
CALL XOPENACC (connection_ID,
               facilities_field_length,
               facilities_field,
               call_user_data_length,
               call_user_data,
               D_bit_services,
               return_code)
```

Parameters

connection_ID (supplied)

Specifies the identifier of the SVC to be accepted. This ID must have been returned on a previous XOPENREC call.

facilities_field_length (supplied)

Specifies the length (from 0 to 63 or 109 bytes) of the X.25 *facilities_field* that the network layer inserts in the Call Accepted packet to be formatted and sent.

Note: If *facilities_field_length* is zero, the *facilities_field* is not used; nor is the *facilities_field* from the VCR used.

facilities_field (supplied)

Specifies the X.25 *facilities_field* (in compliance with the X.25 CCITT recommendation, for example, “fast select”) that the network layer inserts in the Call Accepted packet to be formatted and sent. Ignored if the *facilities_field_length* is 0.

call_user_data_length (supplied)

Specifies the length (from 0 to 128 bytes) of the *call_user_data* to be inserted by the network layer in the Call Accepted packet.

Note: If the *call_user_data_length* parameter is different from 0, the “fast select” facility must be requested.

call_user_data (supplied)

Specifies the user data that the network layer inserts in the Call Accepted packet to be formatted and sent. The “fast select” facility is required. Ignored if the *call_user_data_length* parameter is set to 0.

D_bit_services (supplied)

Specifies that D-bit (delivery confirmation) services can be used during the call (0= services can not be used, 1=services can be used).

This parameter is used either to confirm D_bit service requested by the calling DTE, or to request D_bit service at the calling DTE.

return_code (returned)

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, “X.25 API Reference Information.”

XIT

The X.25 application issues an XIT call to request the network layer to send an Interrupt packet on a previously established or allocated virtual circuit. The interrupt mechanism allows a DTE to transmit from 1 to 32 bytes of data. An Interrupt packet can contain one byte of user data for networks supporting the

XIT

1980 version of the X.25 recommendation, and up to 32 bytes for networks supporting the 1984/88 version. Flow control does not apply to interrupt packets.

Format

```
CALL XIT      (connection_ID,  
              interrupt_data_length,  
              interrupt_data,  
              return_code)
```

Parameters

connection_ID (supplied)

Specifies the identifier of the virtual circuit on which the Interrupt packet is to be sent. It must have been returned on a previous XOPEN, XOPENREC, or XALLOC call.

interrupt_data_length (supplied)

Specifies the length (in bytes) of the interrupt data to be inserted in the Interrupt packet that is to be sent by the network layer. Valid values are:

- 1, to comply with the CCITT 1980 recommendation
- From 1 to 32, to comply with the CCITT 1984 recommendation

interrupt_data (supplied)

Specifies the interrupt data provided by the X.25 application, and to be inserted in the Interrupt packet.

return_code (returned)

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, "X.25 API Reference Information."

XRESET

An XRESET verb is sent to request the network layer to reset a virtual circuit. The circuit is cleared of all data. To ensure that there is no outstanding data from a previous transfer on the virtual circuit, PVCs should normally be reset before starting data transfer. SVCs need not be reset when making or receiving calls because initialization is implicit in the call setup procedure. Once the virtual circuit is reset, the connection ID is still valid, and the virtual circuit is available for data interchange.

If the X.25 API receives a reset request, it automatically confirms the request and discards any data for that virtual circuit. The X.25 API application is informed of the reset indication on its next verb, if the X.25 API was not processing a verb at the time the reset was received. If the X.25 API was processing a verb (for example, during an XSEND), then the current verb completes immediately with an event indication for a reset.

Format

```
CALL XRESET   (connection_id,  
              cause_code,  
              diagnostic_code,  
              return_code)
```

Parameters

connection_ID (supplied)

Specifies the identifier of the virtual circuit to be reset. This connection ID must have been returned on a previous XOPEN, XOPENREC, or XALLOC call.

***cause_code* (supplied)**

Specifies the value of the cause byte to be inserted in the Reset Request packet (the cause byte set by the application must be 0, or greater than or equal to 128, with bit 8 set to 1).

***diagnostic_code* (supplied)**

Specifies the value of the diagnostic byte to be inserted in the Reset Request packet.

***return_code* (returned)**

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, “X.25 API Reference Information.”

XALLOC

An X.25 application issues an XALLOC verb to request the allocation of a PVC. To ensure that there is no outstanding data from a previous transfer on the virtual circuit, PVCs should normally be reset before starting data transfer. See “XRESET” on page 248 for more information.

Figure 13 on page 250 shows how to use this verb.

Format

```
CALL XALLOC    (vcr_name,
                vcr_name_length,
                connection_ID,
                return_code)
```

Parameters

***vcr_name* (supplied)**

Specifies the name of the VCR (1 to 8 alphanumeric characters), as defined by configuration. This parameter is mandatory. Additional XALLOC parameters can be associated with the *vcr_name*.

***vcr_name_length* (supplied)**

Specifies the length in bytes of the name of the VCR.

***connection_ID* (returned)**

Specifies the PVC allocated to the application.

***return_code* (returned)**

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, “X.25 API Reference Information.”

XDEALLOC

```
INTEGER*4 SX:                INTEGER*2 S,SUM
CALL XALLOC (VCR$,           \
             X25.VCR$.NAME.LENGTH, \
             X25.CONNECTION.ID,    \
             X25.RETURN.CODE)
GOSUB I4TOHEX: MESSAGE$ = "XALLOC RCODE: "+ERRFX$: GOSUB PRTMSG
IF X25.RETURN.CODE = 0 THEN BEGIN
  MESSAGE$ = "CONNECTION ID: "+STR$(X25.CONNECTION.ID): GOSUB PRTMSG
! Find an available entry and store the connection ID in the X25.VC.ID Table
  I%=1
  WHILE (X25.VC.ID(I) <> 0)
    I% = I% + 1
  WEND
  X25.VC.ID(I%) = X25.CONNECTION.ID
  X25.VC.INDEX = I%
  MESSAGE$ = "XALLOC VC INDEX= " + STR$(X25.VC.INDEX): GOSUB PRTMSG
! Increment number of currently opened virtual circuit
  X25.NUM.VC = X25.NUM.VC + 1
  RC = -1
  WHILE (RC <> 0)
    CALL XRESET (X25.CONNECTION.ID, \
                 X25.CAUSE.CODE.SEND$, \
                 X25.DIAGNOSTIC.CODE.SEND$, \
                 X25.RETURN.CODE)
    RC = X25.RETURN.CODE
    GOSUB I4TOHEX: MESSAGE$ = "XRESET RCODE: "+ERRFX$: GOSUB PRTMSG
    IF X25.RETURN.CODE = 80B21295h THEN BEGIN
      CALL XRESET (X25.CONNECTION.ID, \
                   X25.EVENT.TYPE, \
                   X25.FACILITY.FIELD.LENGTH, \
                   X25.FACILITY.FIELD.BUFFER$, \
                   X25.DATA.LENGTH, \
                   X25.DATA.BUFFER$, \
                   X25.CAUSE.CODE.REC$, \
                   X25.DIAGNOSTIC.CODE.REC$, \
                   X25.RETURN.CODE)
    ENDIF
    IF RC <> 0 THEN WAIT ;3000
  WEND
! Subroutines
I4TOHEX$:
  ERRFX$ = ""
  FOR S=28 TO 0 STEP -4
    SX = SHIFT(X25.RETURN.CODE,S)
    SUM = SX AND 000FH
    IF SUM>9 THEN SUM = SUM + 55 ELSE SUM = SUM + 48
    ERRFX$ = ERRFX$ + CHR$(SUM)
  NEXT S
  I4TOHEX$ = ERRFX$
RETURN

PRTMSG:
  ....
RETURN
```

Figure 13. XALLOC Verb Example

XDEALLOC

An X.25 application issues an XDEALLOC verb to request the deallocation of a PVC. A valid connection ID is a required parameter. No X.25 packets are sent as a result of a XDEALLOC call.

Format

```
CALL XDEALLOC (connection_ID,
               return_code)
```

Parameters*connection_ID* (**supplied**)

Specifies the identifier of the PVC to be deallocated. This connection ID must have been returned on a previous XALLOC call.

return_code (**returned**)

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, “X.25 API Reference Information.”

XSTO

The XSTO verb provides a mechanism for specifying a maximum verb timeout period (in seconds) for the execution of most X.25 API verbs. A timeout value of -1 causes all verbs except XRECEIVE (receive data) and XOPENREC (receive an incoming call), which have their own timeout values, to wait for a response indefinitely. A security mechanism ensures that an X.25 application can always regain control. If the timer expires before the completion of the verb, the X.25 application should do one of the following:

- If SVC, issue XCLOSE if failing verb was not XOPENREC, XOPEN, or XCLOSE
- If PVC, issue XDEALLOC if failing verb was not XALLOC or DEALLOC

The following timeouts are internal to the X.25 subsystem and take precedence over the XSTO timeout if they occur first:

- XOPEN, XOPENACC (call confirmation timer)
- XRESET (reset confirmation timer)
- XCLOSE (clear confirmation timer)

The XSTO verb does not return any indication of a Clear, Reset, or Interrupt.

Format

```
CALL XSTO      (verb_timeout_seconds,
               return_code)
```

Parameters*verb_timeout_seconds* (**supplied**)

Specifies the length of timeout (in seconds). Valid values are -1 to 32 767, where -1 means to wait indefinitely. The default timeout value is -1.

return_code (**returned**)

Completion return code set by the X.25 API indicating the result of the verb call. For a list of return codes, see Appendix D, “X.25 API Reference Information.”

XWAIT (4680 BASIC Only, Multiple Event Wait)

The XWAIT verb waits for an X.25 event or for data to be available in a pipe. It accepts a 4680 BASIC array of wait IDs and wait types. The verb converts the 4680 BASIC request into the equivalent C call to ADX_CWAIT. The following BASIC code illustrates the definition and use of the XWAIT verb.

ADX_CWAIT

```
!
! XWAIT : Definition
!

SUB XWAIT (NUM.WAITS, WAIT.TIME, WAIT.IDS, WAIT.TYPES, WAIT.EVENT) external

    INTEGER*1 WAIT.TYPES ! Array of wait types. Type of I/O session to wait on:
                        ! 1 : pipe. 2 : Communication Link. 3 : X.25
    INTEGER*2 NUM.WAITS ! Number of I/O session to wait on.
    INTEGER*4 WAIT.TIME, \ Maximum time to wait (ms). Note that 0 = forever
    WAIT.IDS,           - Array of I/O session to wait on.
    WAIT.TYPES          ! A return code indicating which event occurred (e. g.
                        ! 1 : Event on the first session or connection
                        ! 0 : for a timeout, and less than zero for an error)

END SUB

!
! The event arrays
!

DIM X25.WAIT.TYPES (11)
DIM X25.WAIT.IDS   (11)

!
! The XWAIT Call
!

CALL XWAIT (X25.NUM.WAITS, \ ! Pass number of elements in array
            WAIT.TIME,     \ ! Pass maximum time to wait
            X25.WAIT.IDS(1), \ ! Pass address of array
            X25.WAIT.TYPES(1), \ ! Pass address of array
            X25.WAIT.EVENT) !
```

ADX_CWAIT (C and COBOL Only, Multiple Event Wait)

An X.25 application can initiate an ADX_CWAIT. This operation waits for data to be available in a pipe or from the X.25 API. Several waits can be initiated in one call. The first wait to be satisfied is indicated in the return code. The wait terminates if the specified time to wait elapses before any event completes, either in a pipe, on a host communication line, or on an X.25 circuit.

If more than one wait is satisfied at the same time, the one that occurs first in the parameter list is in the return code. As a result, the order of the parameters in the list can be significant. If no time limit is specified, the operating system waits indefinitely until at least one of the waits is satisfied. For pipes, the wait for data to be available in a pipe is satisfied when at least one byte is in the pipe.

The maximum number of waits that can be requested at a time is 30. However, the operating system also uses waits, so this maximum may not be available. If more waits are requested than are available, the X.25 application ends, causing a dump (if the application dump option is enabled).

ADX_CWAIT and the X.25 API

The ADX_CWAIT function enables X.25 API user programs to wait on incoming X.25 data (XRECEIVE) simultaneously with other asynchronous events (for example, pipes). In particular, this function applies to a system that receives messages from both the X.25 link and from other applications through a pipe (for example, an EFT controller task).

The ADX_CWAIT function provides the ability to process information from different sources in a manner that does not involve polling each input during idle periods. However, the function does not provide a true asynchronous API. That is, an X.25 API verb must complete before the application can issue another one.

See the *4690 OS: Programming Guide* for more information about ADX_CWAIT.

X.25 API Message Flow

The following sections provide message flow information for operation of the X.25 API.

Loading the X.25 Driver

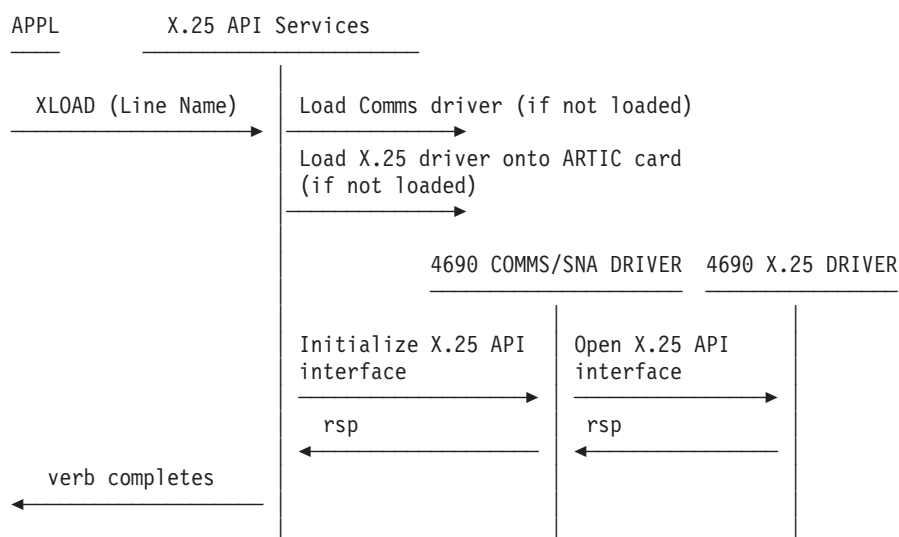


Figure 14. Loading the X.25 Driver for X.25 API Use Only

X.25 API Message Flow — SVC Call Connection

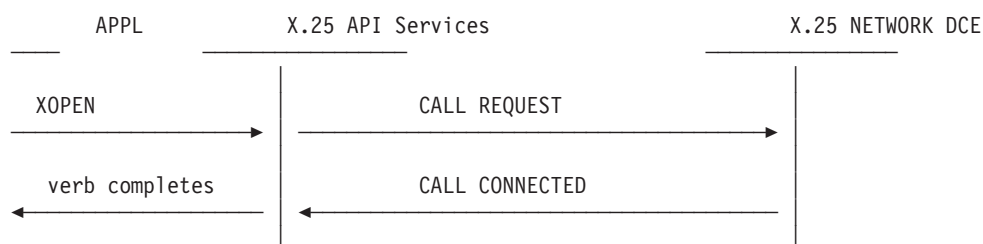


Figure 15. Making an Outgoing Call

ADX_CWAIT

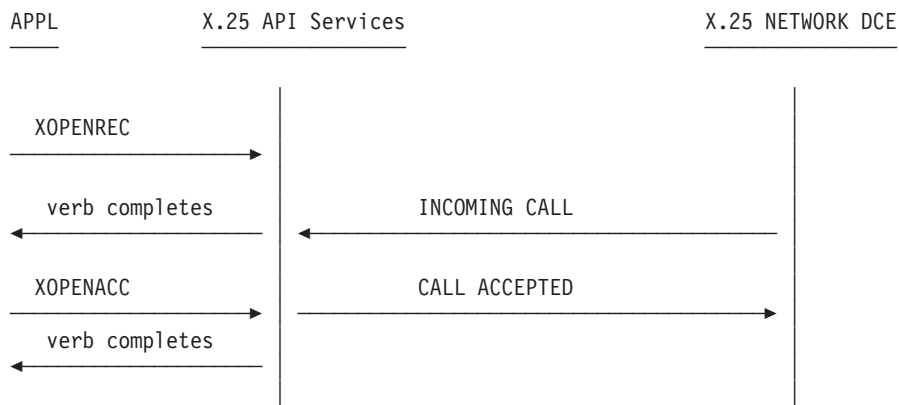


Figure 16. Accepting an Incoming Call

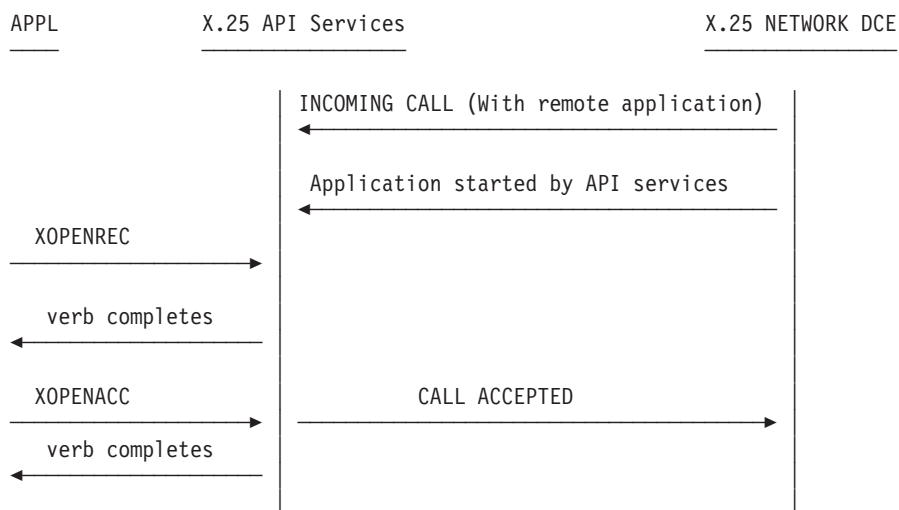
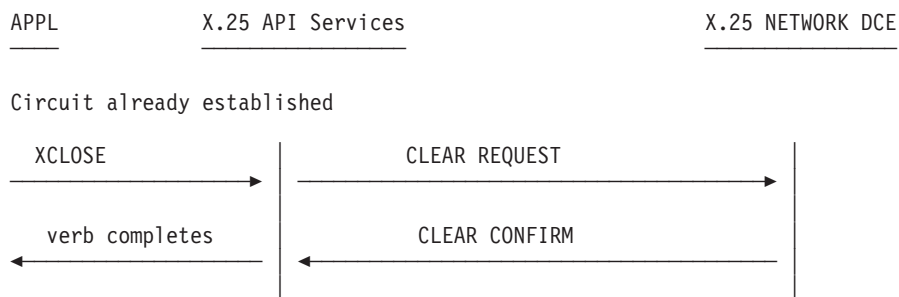


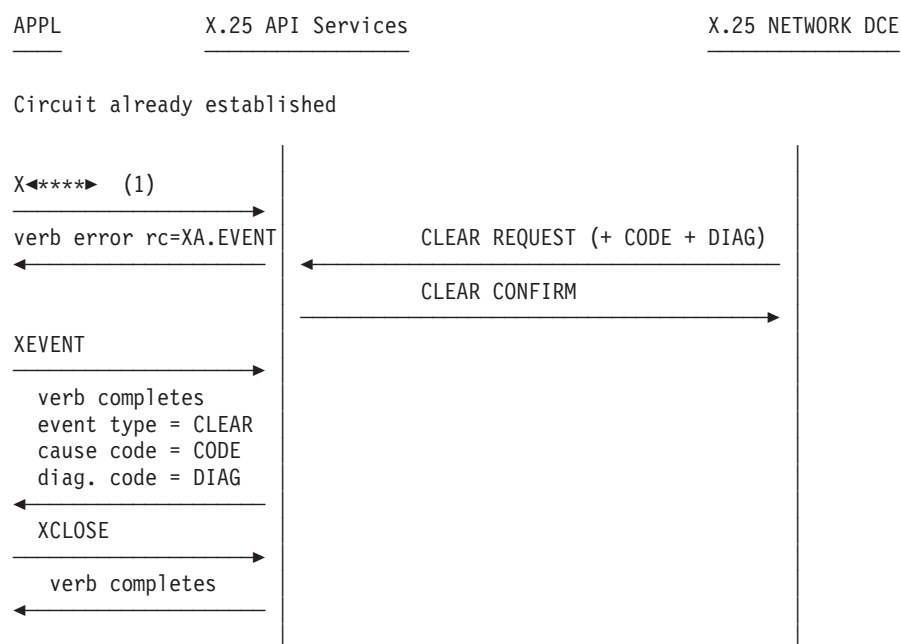
Figure 17. Accepting an Incoming Call with Application Name

X.25 API Message Flow — SVC Call Clearing



Must do XOPEN or XOPENREC to establish circuit again.

Figure 18. Clearing an Active Circuit



Notes:

1. **** Any verb including XOPEN and XOPENREC.
2. Must do XOPEN or XOPENREC to establish circuit again.

Figure 19. Clearing an Already Cleared Circuit

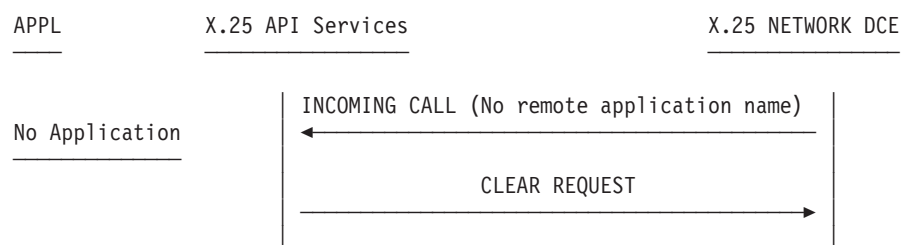
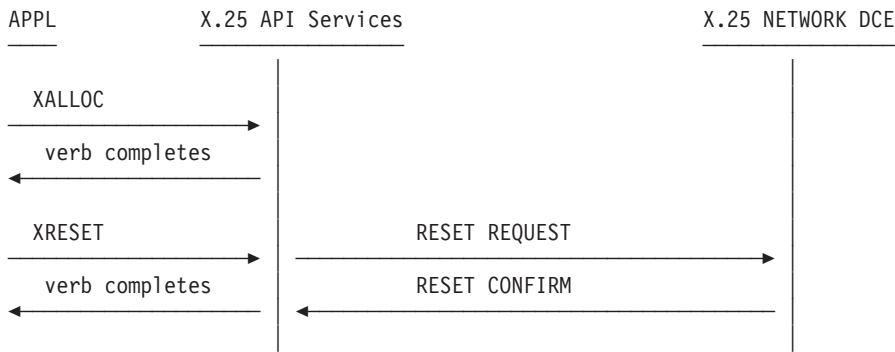


Figure 20. Clearing Call when no XOPENREC waiting

X.25 API Message Flow — PVC Initialization



One appl must now synchronize start of data transmission by periodically sending a "start message" until a "start response" is received.

Other appl now receives data until "start message" is received, then sends "start response" message.

Figure 21. Initializing a PVC Circuit

X.25 API Message Flow — PVC Termination

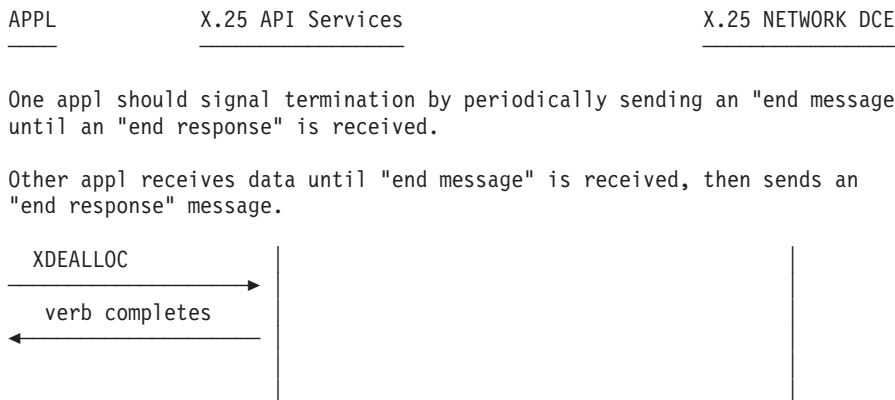
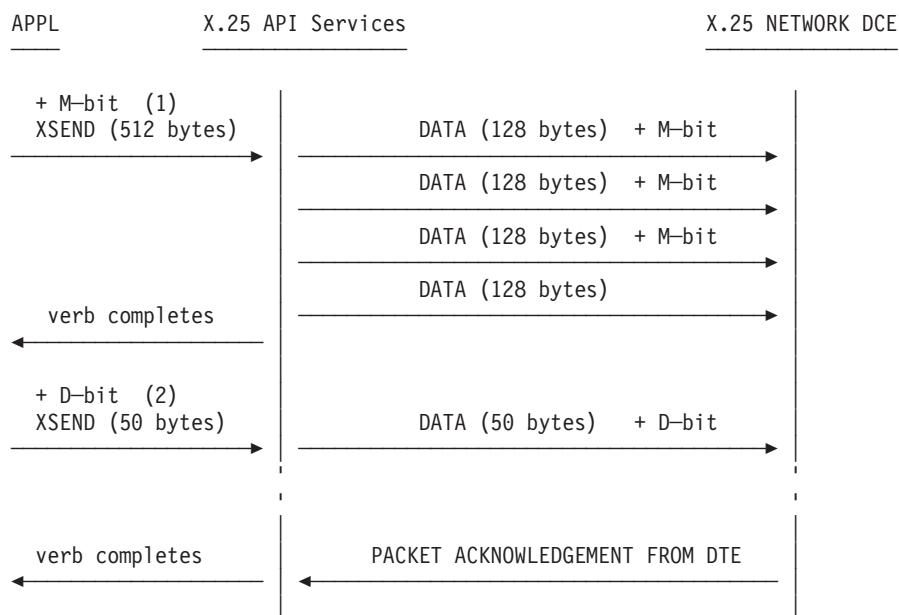


Figure 22. Terminating Application Use of a PVC Circuit

X.25 API Message Flow — Data Transfer



Notes:

1. Amount of data specified is greater than the packet size. All the packets of the sequence, except the last one, will have the M-bit set to 1.
2. D-bit set to 1 requires a delivery confirmation

Figure 23. Sending Data

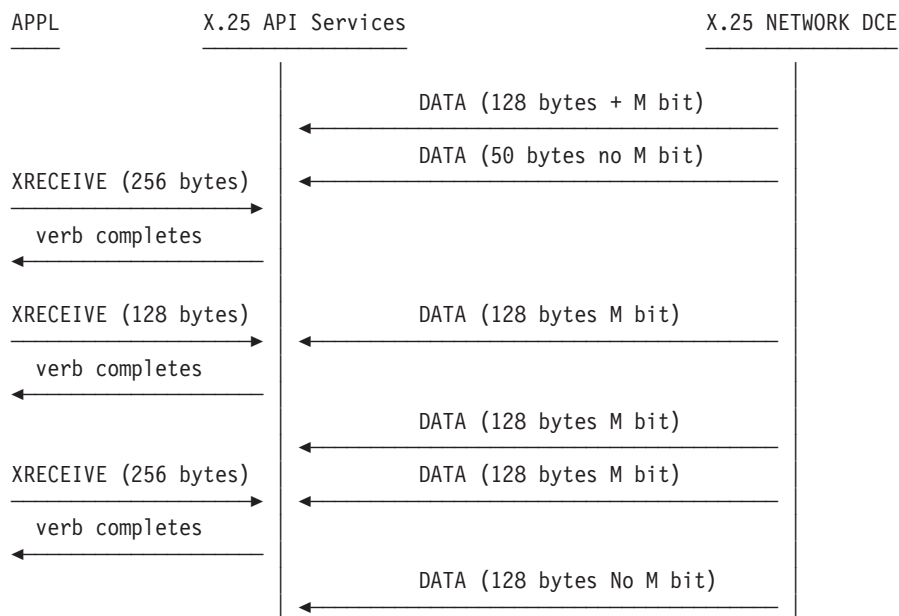


Figure 24. Receiving Data

X.25 API Message Flow — X.25 Events

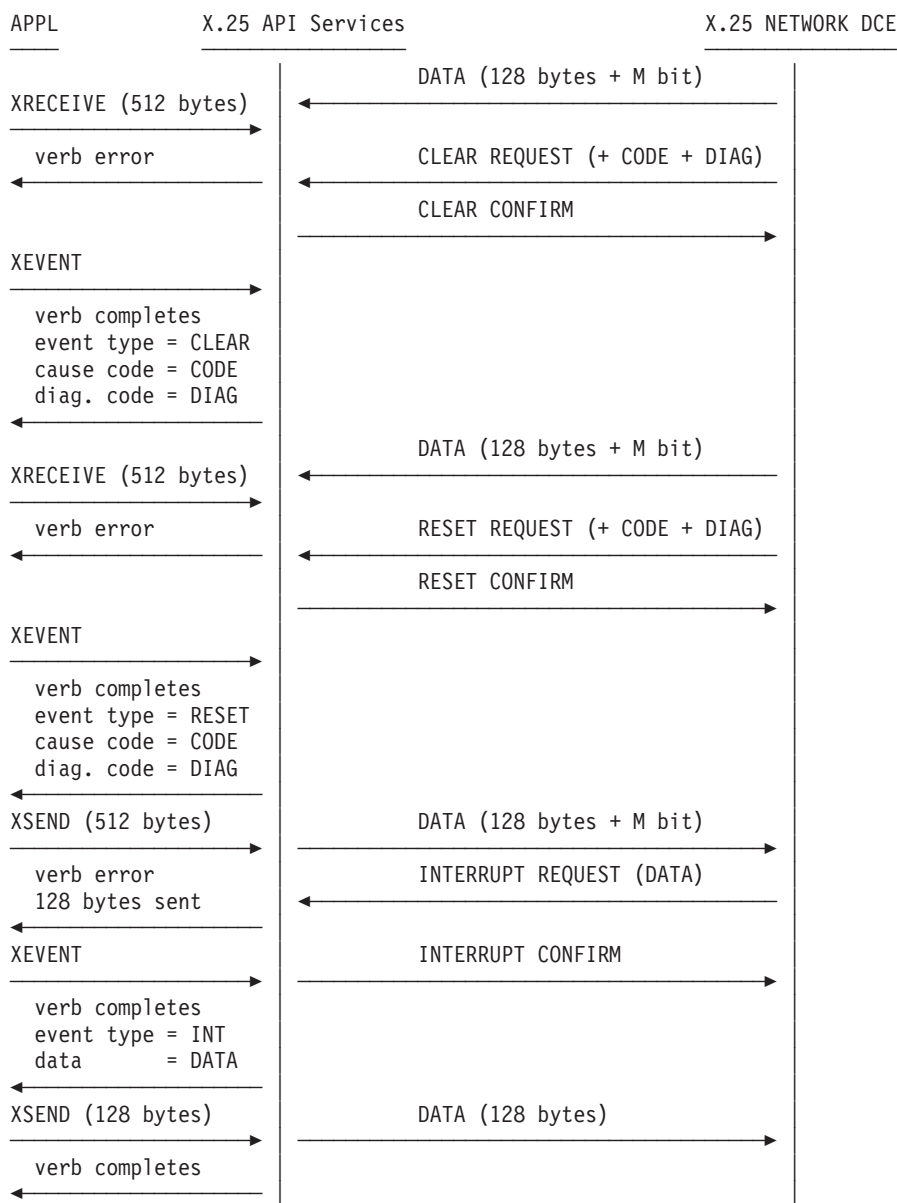


Figure 25. Receiving X.25 Events

Remote Loading of Applications

A calling DTE can request the loading of an application by placing an application logical name into the *call_user_data* field. If no application is waiting to receive the calling DTEs incoming call (that is, there is no matching XOPENREC waiting), the specified application is loaded. This service is not available for SVCs using “fast select”.

The mechanism for passing the 4690 logical name of the application is to use bytes 1 through 8 of the 16-byte *call_user_data* string on the Call Request packet. Byte 0 represents the first byte of call user data, and it is one part of the call acceptance criteria. As a result, it cannot be used to store the remote application name.

Support for remote loading implies that the second byte of *call_user_data* (for calls that are not “fast select”) is reserved. Reserving the second byte indicates that bytes 1 to 8 (of 16) contain an application logical name (in ASCII format). If the second byte is zero, the X.25 API assumes that no name is provided. The remaining 14 bytes (of 16) are available to the application.

The name of the remote application is provided in one of the following ways:

- In the VCR record configuration (for SVCOUT and SVC two-way only)
- A parameter to the XOPEN call

For an application to be loaded, the communications driver must have been initialized prior to the incoming Call Request package. In addition, the following conditions must be satisfied:

- The virtual circuit type is switched.
- The incoming call is not a “fast select” call.
- No application is available to receive the incoming call. For example, no waiting XOPENRECs acceptance criteria match the incoming call.
- The logical name was defined on 4690.
- An application logical name is specified in the *call_user_data* field of the Call Request packet.

If any of these conditions fail, the application is not loaded, and the incoming Call Request is cleared. If there are no waiting applications, the incoming call is cleared with cause code = X'80' and diagnostic code = X'F1'.

If all of the conditions are satisfied, the X.25 API attempts to load the application whose logical name was supplied. If successful, the loaded application performs an XLOAD to initialize the X.25 API, followed by an XOPENREC with acceptance criteria that match the incoming Call Request. The X.25 API next matches the original incoming Call Request to the new XOPENREC verb. As with the existing LU 6.2 interface, the X.25 API loads multiple copies of an application if multiple incoming calls all meet the conditions for loading, and each request specifies the same application logical name. The X.25 API does not attempt to determine that a copy of the application is already running. It determines only that no application is available to receive the call and loads a copy of the requested application. In addition, the X.25 API itself does not perform any translation between EBCDIC and ASCII character sets.

X.25 Return Code and Verb Cross-Reference Table

Table 62. X.25 Verb Return Codes Cross-Reference Table

	X L	O A	D	X S T O	X O P E N	X S E N D	X R E C E I V E	X C L O S E	X E V E N T	X O P E N R E C	X O P E N R E C	X D E A L L O C
XA.OK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
XA.ERR.ADAPTER	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	-
XA.ERR.RESOURCE	Y	-	Y	Y	Y	-	Y	-	Y	Y	Y	Y
XA.ERR.LINE.NOT.LOADED	-	-	Y	-	Y	-	-	-	-	-	-	Y
XA.PVC.INUSE	-	-	-	-	-	-	-	-	-	-	-	Y
XA.NO.SVC.AVAILABLE	-	-	Y	-	Y	-	-	-	-	-	-	-
XA.NOTHING.RECEIVED.IN.DELAY	-	-	-	-	-	Y	-	-	-	Y	-	-
XA.TIMEOUT	-	-	Y	Y	Y	-	Y	-	-	Y	Y	-
XA.IN.PROGRESS	Y	-	-	-	-	-	-	-	-	-	-	-
XA.EVENT	-	-	Y	Y	Y	Y	Y	-	-	Y	Y	-
XA.NO.EVENT	-	-	-	-	-	-	-	Y	-	-	-	-
XA.INVALID.RESPONSE.XSEND	-	-	-	Y	-	-	-	-	-	-	-	-
XA.INVALID.RESPONSE.XCLOSE	-	-	-	-	-	-	Y	-	-	-	-	-
XA.INVALID.RESPONSE.XRESET	-	-	-	-	-	-	-	-	-	-	Y	-
XA.INVALID.RESPONSE.XIT	-	-	-	-	-	-	-	-	-	-	Y	-
XA.TIMEOUT.CALL.CONNECT	-	-	-	-	-	-	-	-	-	Y	-	-
XA.ERR.PARAMETER	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
XA.ERR.CONNECTION.ID	-	-	-	-	Y	Y	Y	Y	Y	-	Y	Y
XA.ERR.TIMEOUT	-	-	-	-	-	-	-	-	-	-	-	-
XA.ERR.D.BIT	-	-	-	Y	-	-	-	-	-	-	-	-
XA.ERR.M.BIT	-	-	-	Y	-	-	-	-	-	-	-	-
XA.RESERVED1	-	-	-	-	-	-	-	-	-	-	-	-
XA.ERR.CALLED.LENGTH	-	-	Y	-	-	-	-	-	-	Y	-	-

Table 62. X.25 Verb Return Codes Cross-Reference Table (continued)

[illegible]

ADX_CWAIT

Chapter 14. Designing an Asynchronous Program

This chapter discusses the considerations you need to know for designing an asynchronous program for use with the operating system. This chapter assumes that you are familiar with asynchronous (ASYNC) communications, and therefore describes only how asynchronous programs should be designed for use with the operating system. See the *Advanced Data Communications for Stores — General Information* guide for more information about asynchronous communications.

ASYNC Record and Character Support

If record mode support is specified at configuration, the end of a message is determined by the receipt of a 1-byte or a 2-byte end-of-record (EOR) sequence as specified during configuration. You can also specify record mode transmission support for cyclic redundancy checking (CRC). If CRC is specified, verify that your application provides the CRC characters for transmission and performs the actual CRC verification upon receipt of the data.

If the record mode is supported, you have the option of having the application provide the EOR sequence in the data transmitted or allowing the ASYNC driver to add the EOR sequence. The EOR sequence is not passed to the application buffer when a read statement is issued, except in CRC mode.

When using record mode, an error code of 80B0000E might be received on a READ statement. This indicates that the current record is larger than the application buffer and that there is more data. In this case, part of the record is in the application buffer. Continue issuing READ statements until the entire record is received.

For character mode, the ASYNC driver does not perform start-of-message or end-of-message processing, but simply transmits the data byte by byte.

ASYNC XON/XOFF Support

The ASYNC driver provides XON/XOFF support if it is specified in the 4690 configuration. When XON/XOFF support is configured, the ASYNC driver sends an XOFF character to the host application when the driver's internal buffers are about to become full. The XOFF character tells the host to stop sending. When the host application has read enough data for the ASYNC driver to again receive data from the host, the ASYNC driver sends an XON character to the host to start sending again. With XON/XOFF configured, the ASYNC driver stops sending to the host when it receives an XOFF character and starts sending again when it receives an XON.

The application can force the ASYNC driver to send an XON or XOFF by using the PUTLONG statement, but only if the ASYNC line is not configured on the Realtime Interface Co-Processor Multiport/2 adapter. If the application forces an XON or XOFF and XON/XOFF support is configured, the ASYNC driver's automatic generation of XON and XOFF characters is overridden. If the ASYNC line is configured to use the Realtime Interface Co-Processor Multiport/2 adapter, the application cannot use the PUTLONG statement to force an XON or XOFF character.

Note: For the application to function properly with or without the Realtime Interface Co-Processor Multiport/2 adapter, Toshiba recommends that you not use the PUTLONG statement to force an XON or XOFF character.

The XON character is X'11', and the XOFF character is X'13'.

ASYNC RS-422 Lines

If an ASYNC line is configured on an RS-422 port of an ARTIC adapter, the ASYNC line automatically uses the RS-422 protocol. RS-422 does not provide any modem controls. Therefore, Toshiba recommends that RS-422 only be used on direct-attach lines. The line is then treated as if the Clear-to-Send (CTS) signal, the Data-Set-Ready (DSR) signal, and the Receive-Line-Signal-Detect (RLSD) signal are always present.

The operating system supports RS-232 ports on the planar board for the attachment of asynchronous devices. These ports can also be used to interface with an Electronic Funds Transfer System (EFTS) network from a sales application.

Starting an ASYNC Line

An ASYNC line is started by using the OPEN LINK ASYNC statement. The OPEN LINK ASYNC statement installs and opens the ASYNC driver, and enables ASYNC communications. See the *4680 BASIC: Language Reference* for statement information.

The BUFFSIZE reserved word of the OPEN LINK ASYNC statement determines the size of the ASYNC driver's internal read and write buffers, and the maximum record size if record I/O is supported. The ASYNC driver's internal read buffer size is the BUFFSIZE value coded on the OPEN LINK ASYNC statement times the number of receive buffers specified during configuration. Write buffer size is the BUFFSIZE value multiplied by the number of transmit buffers specified during configuration. The largest single message the runtime library can process is 518 bytes.

If the ASYNC link is configured for auto-dial, the optional *tele.number* parameter can be coded with a telephone number that overrides the telephone number specified during configuration.

The driver allocates storage and other system resources for use by the driver. Error codes are returned if the required resources cannot be allocated. Other errors that can result from processing the OPEN LINK ASYNC statement are:

- Timeouts while waiting for the signals. The signal timeouts are set at 10 seconds.
- Timeouts waiting for auto-answer or auto-dial. Auto-answer and auto-dial timeout values are specified during configuration.

If a direct-attach (no modem) connection is used, the OPEN LINK ASYNC statement completes successfully when the CTS signal is received. For manual, nonswitched, auto-dial, and auto-answer connection, the following conditions apply:

- Only DSR is required to open.
- CTS is required only for transmission.
- RLSD is required to receive.

If one of the following conditions exists, you must ensure the connection by issuing a 1.5- to 2.0-second wait after the OPEN LINK ASYNC statement completes before attempting a write operation:

- The serial/parallel adapter is a Toshiba 1200-baud internal modem.
- A modem that provides equalization after the answer tone is being used.

Table 63 on page 265 displays the modem control lines that are required to be active for the OPEN statement, the TRANSMIT statement, and the RECEIVE statement. The modem control lines are:

- Clear-to-Send (CTS)
- Data-Set-Ready (DSR)
- Data-Terminal-Ready (DTR)
- Receive-Line-Signal-Detect (RLSD)
- Request-to-Send (RTS)

Table 63. Required Active Modem Control Lines for OPEN, TRANSMIT, and RECEIVE

Connection Types	OPEN	TRANSMIT	RECEIVE
Nonswitched (leased)	DSR	CTS, DSR	DSR, RLSD
Auto-dial	CTS, DSR, RLSD	CTS, DSR, RLSD	CTS, DSR, RLSD
Auto-answer (attention command set compatible modems)	CTS, DSR, RLSD	CTS, DSR, RLSD	CTS, DSR, RLSD
manual	DSR	CTS, DSR	DSR, RLSD
Direct attached (no modem)	CTS	CTS	CTS
Application controlled modem	None	CTS	None
Auto-answer (other modems)	DSR	CTS, DSR	DSR

Note: For auto-answer (non-attention command set compatible) modems, neither DSR nor CTS can be active before the driver has raised RTS and DTR, or the OPEN will fail. The driver raises DTR and waits for DSR. Then the driver raises RTS and waits for CTS. If DSR drops following the OPEN LINK ASYNC statement, the driver lowers DTR.

Reading Data from the Host

To receive data from the ASYNC link, use the READ statement as described in the *4680 BASIC: Language Reference*. When a READ statement is issued, one of the following is returned:

- One record, if record I/O is supported and if a full record was received by the driver.
- Any data received by the ASYNC driver up to the length specified on the *BUFFSIZE* parameter of the OPEN LINK ASYNC statement, if no errors were encountered and record mode is not supported.
- Part of the received data, if an error occurred and record mode is not supported.
- An error code.

If an error occurs on a READ statement during character mode transmission, the length of the data returned to the application from the ASYNC driver is less than the buffer size of the session. When this occurs, the next READ statement to the driver returns the error code of the error encountered. A third READ statement returns any data received after the error. The application must perform any further error processing.

If an error occurs on a read during record mode, an error code is returned and the bad record is ignored.

A read timeout occurs if no data is received by the driver or if a full record is not received by the driver in the time specified during configuration for the READ statement *timeout* value. A value of zero (0) for read timeout causes the ASYNC driver to wait indefinitely.

A good programming practice in application programs is to insert wait periods for the length of time necessary to read an entire ASYNC record. Take into consideration both the line speed and the record length when deciding on an appropriate length of time to wait. (Adding these wait periods can considerably reduce controller load and slightly increase throughput.)

Writing Data to the Host

The WRITE statement and the WRITE FORM statement can be used to transmit data from the application through the ASYNC driver to the host processor. The WRITE statement allows the application to write up to 518 bytes per WRITE statement. The WRITE FORM statement also allows up to 518 bytes to be written. It also allows formatting of the data by the 4680 BASIC runtime subroutines before the ASYNC driver transmits the data. See the *4680 BASIC: Language Reference* for more information on WRITE FORM.

When the ASYNC driver is requested to write, it transfers the data to be transmitted from the application buffer to the ASYNC driver's internal write buffer. Upon completion of this transfer, control is returned to the application. The driver then asynchronously transmits the data from the internal write buffer to the adapter, one byte at a time, until the buffer is empty. The application might receive control before all data is sent to the host. The application can then issue another WRITE statement. If the ASYNC driver's internal buffer has room for the subsequent write data, the data is transferred to the internal buffer and transmitted. Otherwise, control is not returned to the application until enough space is freed in the internal write buffer. Data is written to the ASYNC line, one byte at a time, from the driver's internal buffer as long as all of the following are true:

- The adapter's 1-byte transmission buffer is empty.
- Data is available to write.
- The line active bit is set on in the status byte obtained through the GETLONG statement.
- No outstanding XOFF from the host exists.

If the line drops during transmission, any data remaining in the driver's buffer is sent when the line becomes active again.

If the bits-per-character is less than eight, the significant bits are placed in the rightmost bits of each byte in the application's buffer for a READ statement and are taken from the rightmost bits on a WRITE statement. The following example shows the placement of significant bits in a byte for 6-bit characters, where "X" is a significant bit and "." is a placeholder in the byte:

```
. . X X   X X X
```

Checking the Completion of Read and Write Functions

The WAIT statement can be used in conjunction with the GETLONG statement to wait for asynchronous events to complete or change status. The WAIT statement allows the application to wait on a change in the status flags associated with the ASYNC line and a second, different I/O session number. A change in bits 0, 1, 3, or 6 in the SS byte since the last GETLONG or OPEN LINK ASYNC statement satisfies a wait. When the wait is satisfied, use the EVENT% function to determine the session number that satisfied the wait.

Controlling Use of Read and Write Buffers

The GETLONG statement is used to obtain status information on the ASYNC session and returns an integer in the form FFSSMMPP. Bytes FF, MM, and PP are described in the next section. Byte SS contains status information from the ASYNC driver that can be used by the application for buffer control.

Table 64 illustrates how byte SS represents status information.

Table 64. Status Information Represented by Byte SS

Bit	Value	Status Information
0 (hex 01)	1	Indicates that there is data in the ASYNC driver's internal read buffer available for the application to read in character mode, or indicates there is a full record available in record mode.

Table 64. Status Information Represented by Byte *SS* (continued)

Bit	Value	Status Information
1	1	Indicates that the ASYNC driver's internal write buffer is available for data transfer from the application.
2	0	Reserved or not used.
3	1	Indicates that the communications line is active. For direct attach (no modem), it indicates that the CTS signal is raised. For all other connection types, it indicates that the CTS, DSR, and RLSD signals are raised.
4	1	Indicates that a timeout has occurred.
5	0	Reserved or not used.
6	1	Indicates that the ASYNC driver's internal write buffer is empty. This flag is used to check if all data previously transferred to the internal buffer has been transmitted. Because the application receives control after a write request as soon as the data has been transferred from the application buffer, the application could issue a CLOSE statement on the ASYNC line before all the data was actually transmitted to the host. The application can check this bit to determine if all data has been sent before issuing a CLOSE statement. There might also be cases where the application would intentionally close the link before all data was transmitted in order to purge any remaining data.
7	1	Indicates that an open request has been received by the ASYNC driver.

Note: Refer to "Starting an ASYNC Line" on page 264 for explanations on signals.

Checking and Controlling Modem Interface Signals

Use the GETLONG statement to check the modem status and the modem interface signals. This returns the same integer in the form *FFSSMMPP*. Refer to Table 64 on page 266 for the description of byte *SS*. Bits 0 through 4 of byte *FF* are the modem control bits:

bit 0 Data Terminal Ready
(hex 01)

bit 1 Request to Send

bit 2 Out 1

bit 3 Reserved

bit 4 Loop

bit 5 Used to set bits 0 to 4

bit 6 Used for XON/XOFF

bit 7 Used for XON/XOFF

To modify the modem control bits (bits 0 to 4 of byte *FF*), use the PUTLONG statement with bit 5 turned on and bits 0 to 4 set as they are to be updated.

XON/XOFF, if supported, can be controlled with bits 6 and 7 of byte *FF*. The following describes the meanings of the different settings of bits 6 and 7 for a PUTLONG. GETLONG meanings are in parentheses.

11 XOFF is to be sent (was sent).

01 XON is to be sent (was sent).

00 Ignored (ignored).

10 Ignored (ignored).

Byte *MM* contains the contents of the modem status register. Byte *PP* contains the port number (1 or 2).

Example of an ASYNC Application in 4680 BASIC

The following example is for assisting you in writing a 4680 BASIC application interfacing to ASYNC to communicate with any of the following:

- A host application
- Another 4690 store controller
- A terminal device

Generally, no rigid ASYNC protocol exists. The message interpretation is entirely the responsibility of the application with the exception of configurable options like XON/XOFF, BAUD RATE, and STOP BITS. See the *4680 BASIC Language Reference* for the statement structure. The communications application should contain at a minimum the following:

```
SUB ASYNC.ERR(RETRY.FLAG,OVERLAY$)
! Subroutine for asynchronous errors, such as write errors.
  INTEGER*2  RETRY.FLAG
  STRING     OVERLAY$
! Code to handle asynchronous errors.
END SUB
! Any additional subroutines and functions needed by the application to
! process data between the store controller and connected ASYNC device.
! Main line code - designed based upon the specific requirements.
ON ASYNC ERROR CALL ASYNC.ERR(RETRY.FLAG,OVERLAY.STR$)
ON ERROR GOTO SYN.ERR
! The ASYNC OPEN statement takes one of two forms dependent upon the
! configured line connection type. If the line is not
! an auto-dial line, the OPEN statement is coded as follows, with
! the lowercase entries denoting user specified values.

OPEN LINK ASYNC link.name AS link.number BUFFSIZE buffersize

! The second form of the ASYNC OPEN statement allows the inclusion
! of a telephone number for an auto-dial line. This number
! overrides the configured telephone number.

OPEN LINK ASYNC link.name, tele.no AS link.number BUFFSIZE buffersize
```

If the OPEN LINK ASYNC statement was successful, the code used for controlling the information flow depends on application requirements. Because the protocol is user implemented, the methodology is also a user responsibility.

See the *4680 BASIC Language Reference* for a description of the commands that can be issued and their functions.

Stopping an ASYNC Line

An ASYNC line is stopped by coding a CLOSE statement with the number of the link to be closed as specified in the OPEN statement. Closing the ASYNC line causes data in buffers to be purged. Use the GETLONG statement as previously described to check if data remains in the buffers to prevent data loss prior to issuing a CLOSE statement.

Chapter 15. Designing Applications with C and COBOL Languages

This chapter contains coding guidelines and restrictions that you should consider when you are writing a program in a language other than 4680 BASIC to run under the operating system. It contains specific information about programming applications in C and COBOL.

This chapter is not intended to be an instructional aid for programming in C and COBOL. This chapter assumes that you have a working knowledge of C and COBOL, and that you are familiar with how to write a program in one of these languages.

The interfaces described in this chapter provide access to many parts of the operating system. This chapter explains how to use these interfaces.

Open Communications Link

ADX_COPEN_LINK opens a communications link.

16-Bit C Interface

```
long          lnum;
char          *linkname,*optinfo;
unsigned int   linktype,bufferize;
```

```
ADX_COPEN_LINK ( &lnum,linkname,optinfo,linktype,bufferize );
```

32-Bit C Interface

```
long          lnum;
char          *linkname,*optinfo;
unsigned short linktype,bufferize;
```

```
ADX_COPEN_LINK ( &lnum,linkname,optinfo,linktype,bufferize );
```

COBOL Interface

```
77 LNUM          PIC S9(8)          USAGE IS COMP-5.
77 LINKNAME      PIC X(9)           USAGE IS DISPLAY.
77 OPTINFO       PIC X(34)          USAGE IS DISPLAY.
77 LINKTYPE      PIC 9(4)           USAGE IS COMP-5.
77 BUFFERIZE     PIC 9(4)           USAGE IS COMP-5.
```

```
CALL "ADX_COPEN_LINK" USING LNUM,
BY REFERENCE LINKNAME, BY REFERENCE OPTINFO,
BY VALUE LINKTYPE, BY VALUE BUFFERIZE.
```

Parameters:

linkname

Link name specified during configuration. This parameter is required and it must be terminated with a blank or null. The maximum length is nine characters, including the blank or null.

optinfo

Optional information for link. A telephone number can be specified for ASYNC or a host ID can be specified for SNA. If optional information is not used, *optinfo* must be blank or null.

- Optional telephone number (ASYNC only). Overrides the telephone number specified during system configuration. It must terminated with a blank or null. May be up to 34 characters, including the null or blank.

- Optional host ID (SNA only). Overrides the host ID specified during system configuration. Must be terminated with a blank or null. For SNA, it can be up to 7 characters, including null or blank.

linktype

0 ASYNC
2 SNA

buffsize

Size of communications buffer. For ASYNC, it can be up to 518 bytes. SNA is not specified because the buffer size is set internally.

Inum Return code. It is the link number or an error code if an error occurred.

Error Code: See the *4690 OS: Messages Guide* for system errors.

Open SNA Session

ADX_COPEN_SESS opens a session for an SNA link.

16-Bit C Interface

```
long          snum, lnum;
char          *sessname, *bindbuff;
```

```
ADX_COPEN_SESS ( &snum, lnum, sessname, bindbuff);
```

32-Bit C Interface

```
long          snum, lnum;
char          *sessname, *bindbuff;
```

```
ADX_COPEN_SESS ( &snum, lnum, sessname, bindbuff);
```

COBOL Interface

```
77 SNUM          PIC S9(8)          USAGE IS COMP-5.
77 LNUM          PIC S9(8)          USAGE IS COMP-5.
77 SESSNAME      PIC X(9)           USAGE IS POINTER.
77 BINDBUFF      PIC X(256)         USAGE IS POINTER.
```

```
CALL "ADX_COPEN_SESS" USING SNUM,
BY VALUE LNUM, BY VALUE SESSNAME, BY VALUE BINDBUFF
```

Parameters:

Inum Link number returned from the open link operation. This parameter is required.

sessname

Pointer to a null or blank terminated ASCII string containing the session name as defined for the SNA link during system configuration. The maximum length is 9 characters, including the null or blank.

bindbuff

Pointer to the buffer to receive the bind information. Buffer must be 256 bytes.

snum Return code. It is the session number or an error code if an error occurred.

The BIND buffer receives the BIND information returned on completion of the open session operation. If the BIND buffer address is zero (null), the open completes as soon as all control blocks are allocated. Issuing a read to the session then obtains the BIND request sent from the host. This method of operation should be used by application programs that do not send an INIT-SELF command to the host, if the host has not yet issued the BIND request.

Error Code

See the *4690 OS: Messages Guide* for system errors.

Close Communications Link or Session

ADX_CCLOSE_LS closes a link or a session. All sessions must be closed before the link can be closed.

16-Bit C Interface

```
long                lnum,ret;
```

```
ADX_CCLOSE_LS ( &ret,lnum);
```

32-Bit C Interface

```
long                lnum,ret;
```

```
ADX_CCLOSE_LS ( &ret,lnum);
```

COBOL Interface

```
77  RET              PIC S9(8)          USAGE IS COMP-5.  
77  LSNM             PIC S9(8)          USAGE IS COMP-5.
```

```
CALL "ADX_CCLOSE_LS" USING RET,  
BY VALUE LSNM.
```

Parameters

lnum Link number returned from the open link operation or session number returned from the open session operation. This parameter is required.

ret Return code. It contains an error code if an error occurred.

Error Code

See the *4690 OS: Messages Guide* for system errors.

Read Data from Link or Session

ADX_CREAD_HOST reads data from an SNA session or from an ASYNC link .

16-Bit C Interface

```
long                lnum,buffsize,nbytes;  
char                *buffadr;
```

```
ADX_CREAD_HOST ( &nbytes,lnum,buffadr,buffsize);
```

32-Bit C Interface

```
long                lnum,buffsize,nbytes;  
char                *buffadr;
```

```
ADX_CREAD_HOST ( &nbytes,lnum,buffadr,buffsize);
```

COBOL Interface

```
77  NBYTES           PIC S9(8)          USAGE IS COMP-5.  
77  LSNM             PIC S9(8)          USAGE IS COMP-5.  
77  BUFFADR          USAGE IS POINTER.  
77  BUFFSIZE         PIC 9(8)           USAGE IS COMP-5.
```

```
CALL "ADX_CREAD_HOST" USING NBYTES,
BY VALUE LSNUM, BY VALUE BUFFADR, BY VALUE BUFFSIZE.
```

Parameters:

lsnum Link number returned by the open link operation (ASYNC) or session number returned by the open session operation (SNA). This parameter is required.

buffadr

Address of buffer in which to place data.

buffsize

- 526 if reading from a session
- User-specified if reading from ASYNC link

nbytes

Return code. It contains the number of bytes read or an error code if an error occurred.

Error Code

See the *4690 OS: Messages Guide* for system errors.

Write Data to Link or Session

ADX_CWRITE_HOST writes data to an SNA session or to an ASYNC link.

16-Bit C Interface

```
long          lsnum,buffsize,nbytes;
char          *buffadr;
ADX_CWRITE_HOST ( &nbytes,lsnum,buffadr,buffsize);
```

32-Bit C Interface

```
long          lsnum,buffsize,nbytes;
char          *buffadr;
ADX_CWRITE_HOST ( &nbytes,lsnum,buffadr,buffsize);
```

COBOL Interface

77	NBYTES	PIC S9(8)	USAGE IS COMP-5.
77	LSNUM	PIC S9(8)	USAGE IS COMP-5.
77	BUFFADR		USAGE IS POINTER.
77	BUFFSIZE	PIC 9(8)	USAGE IS COMP-5.

```
CALL "ADX_CWRITE_HOST" USING NBYTES,
BY VALUE LSNUM, BY VALUE BUFFADR, BY VALUE BUFFSIZE.
```

Parameters:

lsnum Link number returned by the open link operation (ASYNC) or session number returned by the open session operation (SNA). This parameter is required.

buffadr

Address of buffer containing data to be written.

buffsize

Number of bytes to write. May not exceed 518 bytes for any type of write.

nbytes

Return code. If the write was successful, *nbytes* is zero for ASYNC. For SNA, *nbytes* equals the number of bytes written. An error code is returned if an error occurred.

Error Code

See the *4690 OS: Messages Guide* for system errors.

Obtain Status from Link or Session

ADX_CGET_STAT obtains information from an open link or SNA session. When checking status, the bit indicating data-ready-to-read is set when one byte is received. Multiple reads might be required to receive the data. The timer function can also be used to wait before reading.

16-Bit C Interface

```
long          lnum,ret;
char          *buffadr;
ADX_CGET_STAT ( &ret,lnum,buffadr);
```

32-Bit C Interface

```
long          lnum,ret;
char          *buffadr;
ADX_CGET_STAT ( &ret,lnum,buffadr);
```

COBOL Interface

```
77 RET          PIC S9(8)          USAGE IS COMP-5.
77 LSNUM         PIC S9(8)          USAGE IS COMP-5.
77 BUFFADR       USAGE IS POINTER.
```

```
CALL "ADX_CGET_STAT" USING RET,
BY VALUE LSNUM, BY VALUE BUFFADR.
```

Parameters:

lnum Link number returned by the open link operation (ASYNC) or session number returned by the open session operation (SNA). This parameter is required.

buffadr

Address of buffer in which to place status information. Buffer must be 8 bytes. See “Buffer for ASYNC Status” and “ASYNC Status Flags.” For SNA sessions see “Buffer for SNA Session” on page 274 and “SNA Status Flags” on page 274.

ret Return code. It contains an error code if an error occurred.

Error Code

See the *4690 OS: Messages Guide* for system errors.

Buffer for ASYNC Status

The buffer comes in the following format:

BUFFER FOR ASYNC STATUS	
Byte	
0	Reserved
4	Status Flags and Data

ASYNC Status Flags

The tables in this section describe the status flags and data.

Table 65. ASYNC Status Flags for Byte 4

Bit	Set To	Indicates
0	1	Data terminal ready (DTR)

Table 65. ASYNC Status Flags for Byte 4 (continued)

Bit	Set To	Indicates
1	1	Request to send (RTS)
2	1	Out 1
3	1	Reserved
4	1	Loop
5	1	The modem control register was set with SEND_REQ
6	1	A send XOFF or XON was set with SEND_REQ
7	0	A send XON was set with SEND_REQ
7	0	A send XOFF was set with SEND_REQ

Table 66. ASYNC Status Flags for Byte 5

Bit	Set To	Indicates
0	1	Data is ready to be read
1	1	Write buffer available
2	1	XOFF active
3	1	Link active (if directly attached, CTS is active)
4	1	Timeout has occurred
5	1	PC XOFF active
6	1	Write buffer empty
7	1	OPEN request received

Table 67. ASYNC Status Flags for Byte 6

Modem Status Register

Table 68. ASYNC Status Flags for Byte 7

Port Number (1 or 2)

Buffer for SNA Session

The buffer comes in the following format:

BUFFER FOR SNA SESSION

Byte

0	Reserved	
4	0	Flags

SNA Status Flags

The tables in this section describe the status flags and data.

Table 69. SNA Status Flags for Byte 5

Bit	Set To	Indicates
0	1	Receipt of an ACTLU request
1	1	Receipt of a BIND request
2	1	Sent an INIT_SELF
3	1	Sent a TERM_SELF

Table 69. SNA Status Flags for Byte 5 (continued)

Bit	Set To	Indicates
4	1	SNA session is active
5	1	Receipt of an UNBIND request
6	1	Receipt of a pacing response
7	1	Receipt of a pacing request

Table 70. SNA Status Flags for Byte 6

Bit	Set To	Indicates
0	1	LINK/SESSION error has occurred
1	1	BIND response is required
2	1	Driver is waiting for the link or session to close
3	1	Open session is in progress
4	1	Close session is in progress
5	1	SNA session is open
6	1	Data is available to be read
7	1	Write buffer is free

Table 71. SNA Status Flags for Byte 7

Bit	Set To	Indicates
0	1	Receipt of an STSN
1	1	Receipt of a CLEAR
2	1	Receipt of a QUIESCE
3	1	Receipt of an SDT
4	1	A read of SNA session is outstanding
5	1	A write of SNA session is outstanding
6		Reserved
7		Reserved

Buffer for SNA Link

The buffer comes in the following format:

BUFFER FOR SNA LINK		
Byte		
0	Reserved	
4	Flags	Not Used

SNA Link Flags

The tables in this section describe the status flags and data.

Table 72. SNA Link Flags for Byte 4

Bit	Set To	Indicates
0	1	ACTPU for the link has been received (sessions can now be activated)
1	1	DACTPU request has been received from the host (all applications must close any sessions and links that have been active)

Table 72. SNA Link Flags for Byte 4 (continued)

Bit	Set To	Indicates
2	1	Active host link has been established
3	1	Link error has been detected (the link must be closed until the error is corrected)
4	1	C & SM support is active on this link
5	1	OPEN LINK request is pending
6		Reserved
7		Reserved

Table 73. SNA Link Flags for Byte 5

Bit	Set To	Indicates
0	1	Read to the SDLC support is outstanding
1		Reserved
2		Reserved
3		Reserved
4		Reserved
5		Reserved
6		Reserved
7		Reserved

Requests to the Driver

ADX_CSEND_REQ sends requests to the SNA and ASYNC drivers.

16-Bit C Interface

```
long          lsnnum,ret;
char          *buffadr;
ADX_CSEND_REQ ( &ret,lsnnum,buffadr);
```

32-Bit C Interface

```
long          lsnnum,ret;
char          *buffadr;
ADX_CSEND_REQ ( &ret,lsnnum,buffadr);
```

COBOL Interface

```
77 RET          PIC S9(8)          USAGE IS COMP-5.
77 LSNUM        PIC S9(8)          USAGE IS COMP-5.
77 BUFFADR      USAGE IS POINTER.
```

```
CALL "ADX_CSEND_REQ" USING RET,
BY VALUE LSNUM, BY VALUE BUFFADR.
```

Parameters:

lsnnum Link number returned by the open link operation (ASYNC only) or session number returned by the open session operation (SNA). This parameter is required.

buffadr

Address of buffer that contains request data. Buffer must be 8 bytes. See “Set Buffer for ASYNC

Status” and “ASYNC Status Flags.” For SNA sessions, see “Set Buffer Area for SNA Status” and “SNA Status Flags.”

ret Return code. It contains an error code if an error occurred.

Error Code

See the *4690 OS: Messages Guide* for system errors.

Set Buffer for ASYNC Status

The buffer comes in the following format:

SET BUFFER FOR ASYNC STATUS		
Byte		
0	Reserved	
4	Flags	Reserved

ASYNC Status Flags

Table 74 describes the status flags and data.

Table 74. ASYNC Status Flags for Byte 4

Bit	Set To	Indicates
0	1	Set data terminal to ready (DTR)
1	1	Set a request to send (RTS)
2	1	Set out 1
3	1	Reserved
4	1	Set the loop
5	1	Set the modem control register
6	1	Send XOFF or XON (see bit 7)
7	0	Send XON
7	1	Send XOFF

Set Buffer Area for SNA Status

The buffer comes in the following format:

SET BUFFER AREA FOR SNA STATUS		
Byte		
0	Number of Bytes Reserved	
4	Flags	Reserved

SNA Status Flags

Table 75 describes the status flags and data.

Table 75. SNA Status Flags for Byte 4

Bit	Set To	Indicates
0	1	Send INIT_SELF
1	1	Send TERM_SELF
2	1	Send RQR
3		Reserved
4		Reserved

Table 75. SNA Status Flags for Byte 4 (continued)

Bit	Set To	Indicates
5		Reserved
6		Reserved
7		Reserved

Appendix A. Using HCP with SNA Request/Response Units

This appendix discusses SNA request/response units used for host connection 2.0 and HCP sequences. The following RUs are sent from the HCP to the HCP LU in the store controller:

- Bind
- Cancel
- Chase
- Clear
- Data
- Data and Control
- LUs
- QEC
- Release Quiesce
- STSN
- Shutdown
- SDT
- Unbind

Note: For all path information units (PIUs), the HCP LU address is X'01'.

The following request/response units are sent from the HCP LU in the store controller to the host processor:

- Cancel
- Complete
- Data

SNA TH/RH/RU Contents

The tables in this section list the transmission header (TH) profiles and the request/response header (RH) profiles used in the host processor to store controller communications. The TH and RH profile settings for Command Requests to and from the Host Program are also shown.

Transmission Header

Figure 26 illustrates the TH profiles used in host processor to store controller communications.

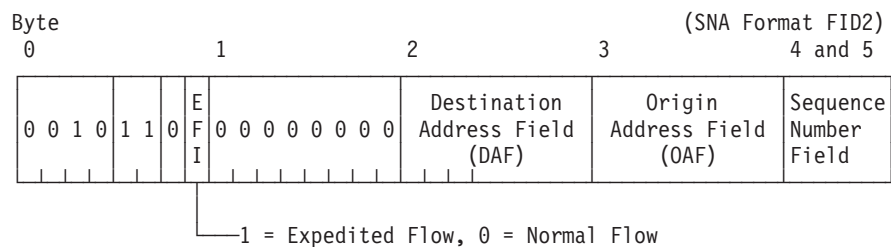


Figure 26. TH Profiles Used in Host Processor to Store Controller Communications

Request/Response Header

Figure 27 on page 280 illustrates the RH profiles used in host processor to store controller communications.

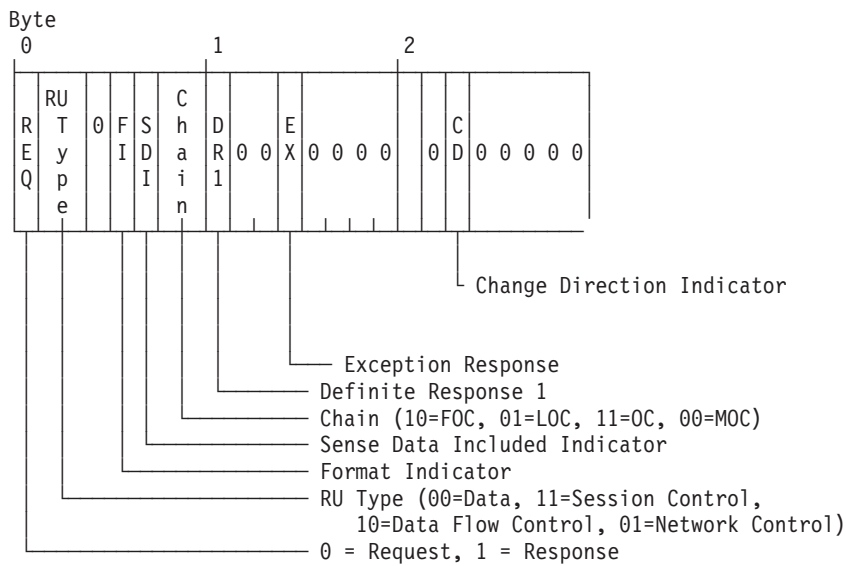


Figure 27. RH Profiles Used in Host Processor to Store Controller Communications

TH and RH Profiles for Command Requests

Figure 28 defines command requests sent by the host program.

Request Name	Transmission Header (TH)						Request Header (RH)			Request Unit (RU)									RU Len
	0	1	2	3	4	5	0	1	2	0	1	2	3	4	5	6	7	8	
Activate Physical Unit	2D	00	00	00	00	00	6B	80	00	11	01	00	00	00	00	00	00	00	9
Deactivate Physical Unit	2D	00	00	00	00	00	6B	80	00	12	01								2
Activate Logical Unit	2D	00	aa	00	00	00	6B	80	00	0D	01	00							3
Deactivate Logical Unit	2D	00	aa	00	00	00	6B	80	00	0E									1
Bind (HCP)	2D	00	aa	bb	00	00	6B	80	00	See note.									37
Unbind	2D	00	aa	bb	00	00	6B	80	00	32	01								2
Start data traffic	2D	00	aa	bb	00	00	6B	80	00	A0									1
Clear	2D	00	aa	bb	00	00	6B	80	00	A1									1
Cancel	2C	00	aa	bb	Seq #		43	80	z0	83									1
FM command	2C	00	aa	bb	Seq #		0x	w0	z0	HCP cmd									<256
FMDATA	2C	00	aa	bb	Seq #		0y	w0	z0	HCP pgm data									<256

Figure 28. Command Requests from the Host Program

Note: See “HCP BIND Request Unit Format” on page 283.

The variables in Figure 28 on page 280 are defined as follows:

n	Name of host application program (primary network addressable unit (NAU) name).
aa	DAF
bb	OAF (X'01' is HCP).
w	X'8' if only chain, or last of chain (no data RUs to follow). In any other condition, w is X'9'.
x	X'B' if only chain. X'A' if first of chain.
y	X'0' if middle of chain. X'1' if last of chain.
z	X'2' if change direction required.
z	X'A' if change direction required. In any other condition, z is X'0'.

Figure 29 on page 282 provides information for diagnostic purposes.

Positive Response (+RSP) To	Transmission Header (TH)						Response Header (RH)			Response Unit (RU)										RU Len
	0	1	2	3	4	5	0	1	2	0	1	2	3	4	5	6	7	8	9	
Activate Physical Unit	2D	00	00	00	00	00	EB	80	00	11	01	F3	F6	F5	F1	F0	F0	F0	F0	10
Deactivate Physical Unit	2D	00	00	00	00	00	EB	80	00	12										1
Activate Logical Unit	2D	00	00	aa	00	00	EB	80	00	0D	01									2
Deactivate Logical Unit	2D	00	00	aa	00	00	EB	80	00	0E										1
Bind (HCP)	2D	00	bb	aa	00	00	EB	80	00	31										1
Unbind	2D	00	bb	aa	00	00	EB	80	00	32										1
Start data traffic	2D	00	bb	aa	00	00	EB	80	00	A0										1
Clear	2D	00	bb	aa	00	00	EB	80	00	A1										1
Cancel	2C	00	bb	aa	Seq #		C3	80	z0	83										1

Negative Response (-RSP) To	Transmission Header (TH)						Response Header (RH)			Response Unit (RU)										RU Len
	0	1	2	3	4	5	0	1	2	0	1	2	3	4	5	6	7	8	9	
Activate Physical Unit	2D	00	00	00	00	00	EF	90	00	ss	ss	00	00	11						5
Deactivate Physical Unit	2D	00	00	00	00	00	EF	90	00	ss	ss	00	00	12						5
Activate Logical Unit	2D	00	00	aa	00	00	EF	90	00	ss	ss	00	00	0D						5
Deactivate Logical Unit	2D	00	00	aa	00	00	EF	90	00	ss	ss	00	00	0E						5
Bind (HCP)	2D	00	bb	aa	00	00	EF	90	00	ss	ss	00	00	31						5
Unbind	2D	00	bb	aa	00	00	EF	90	00	ss	ss	00	00	32						5
Start data traffic	2D	00	bb	aa	00	00	EF	90	00	ss	ss	00	00	A0						5
Clear	2D	00	bb	aa	00	00	EF	90	00	ss	ss	00	00	A1						5
Bid	2C	00	bb	aa	Seq #		C7	90	00	ss	ss	uu	uu	C8						5
Cancel	2C	00	bb	aa	Seq #		C7	90	00	ss	ss	uu	uu	83						5
HCP Command	2C	00	bb	aa	Seq #		8F	90	00	ss	ss	uu	uu	cc	cc	cc				7
HCP Data (HOST/ISC)	2C	00	bb	aa	Seq #		8F	90	00	ss	ss	uu	uu							4

Figure 29. Command Responses to the Host Program

The variables in Figure 29 are defined as follows:

- aa** OAF (X'01' is HCP)
- bb** DAF (X'01' through X'FF' is a customer-specifiable option)
- s** 2-byte system sense code
- u** 2-byte HCP sense code (see the *4690 OS: Messages Guide*)

Figure 30 on page 283 provides information for diagnostic purposes.

Request Name	Transmission Header (TH)						Request Header (RH)			Request Unit (RU)								RU Len
	0	1	2	3	4	5	0	1	2	0	1	2	3	4	5	6	7	
Cancel	2C	00	bb	aa	Seq #		43	80	20	83								1
HCP Command	2C	00	bb	aa	Seq #		0x	w0	z0	HCP cmd								<256
HCP Data	2C	00	bb	aa	Seq #		0y	w0	z0	HCP data								<256

Figure 30. Command Requests to the Host Program

The variables in Figure 30 are defined as follows:

- aa** OAF for requests/responses from the store controller to the host (X'01' is HCP).
- bb** DAF for requests/responses from the store controller to the host (X'01' through X'FF' is a customer-specifiable option).
- n** ACB name
- w** X'8' if only or if last of chain. In any other condition, w is X'9'.
- x** X'B' if only chain. X'A' if first of chain.
- y** X'0' if middle of chain. X'1' if last of chain.
- z** X'2' if change direction required. In any other condition, z is X'0'.

HCP BIND Request Unit Format

Table 76 describes the formats of the HCP BIND request.

Table 76. Formats of the HCP BIND Request Unit

Byte	Contents	Description
0	X'31'	Request Code
1, bits 0–3	X'0'	BIND Format
1, bits 4–7	X'1'	Type = COLD
2	X'04'	FM Profile
3	X'04'	TS Profile
4, bit 0	b1	Multiple RU chains allowed from primary NAU
4, bit 1	b0	Immediate request mode on primary to secondary session
4, bit 2–3	b11	Definite or exception response
4, bit 4–5		Reserved
4, bit 6	b0	Compression not used on requests from primary
4, bit 7	b0	EB not sent from primary
5, bit 0	b1	Multiple RU chains allowed from secondary NAU
5, bit 1	b0	Immediate request mode on secondary to primary half session
5, bits 2–3	b11	Definite or exception response
5, bits 4–5		Reserved
5, bit 6	b0	Compression must not be used
5, bit 7	b0	Secondary cannot send EB

Table 76. Formats of the HCP BIND Request Unit (continued)

Byte	Contents	Description
6, bit 0		Reserved
6, bit 1	b1	FM headers are allowed (bit on indicates HCP Command)
6, bit 2	b0	Brackets will not be used
6, bit 3	b0	Reserved because brackets are not supported
6, bit 4	b0	Alternative code cannot be used
6, bits 5–7		Reserved
7, bits 0–1	b00 b01 * b10 *	Full Duplex (SDLC only) HDX Contention (SDLC) HDX Flip Flop (SDLC)
7, bit 2	b0	Contention loser is responsible for recovery
7, bit 3	b0	Secondary is contention winner and primary is contention loser
7, bits 4–6		Reserved
7, bit 7	b0	Flip Flop reset states
8, bit 0	b0 b1	Inbound pacing one stage Inbound pacing two stages
8, bits 2–7		Inbound pacing count is not to exceed seven, if pacing is wanted. Specify zero, if pacing is not wanted.
9, bits 0–1		Reserved
9, bits 2–7	X'nn'	Outbound pacing receive count equal to zero, if the system definition default is wanted or a value not to exceed seven.
10, bit 0	b1	Indicates maximum RU size sent by the HCP is specified
10, bits 1–7	X'85'	Specifies a maximum RU size of 256 can be sent by the HCP
11, bits 0–7		Defined the same as byte 10 but for RU. Received by the HCP.
12, bits 0–7		As appropriate
13, bits 0–7		As appropriate
14–26	X'00'	
27	X'nn'	Length of primary NAU name
28–35	X'n..n'	n..n is the name of the primary LU associated with the HCP session

Note:

- * For these protocols, all requests are exception response only RUs.
- VTAM® software users only: ACF/VTAM® software supports pacing. Be sure to verify VTAM support before specifying pacing.

When the BIND request is received, the fields containing the contents shown in Table 76 on page 283 are all checked. The following error responses can be sent for the BIND request:

Session Limit Exceeded (X'0805')

If the HCP in the operating system is in session with another LU and a BIND is received, the BIND is rejected with this error code.

Function Active (X'0815')

This error code is sent in response to a BIND received for a session that is already bound.

Invalid Session Parameter (X'0821')

This error code is sent if the BIND parameters are not set as shown in Table 76 on page 283.

Appendix B. HCP Communications with SDLC Protocols

This appendix contains protocol examples that can be observed when the host is communicating with the HCP in a synchronous data link control (SDLC) environment.

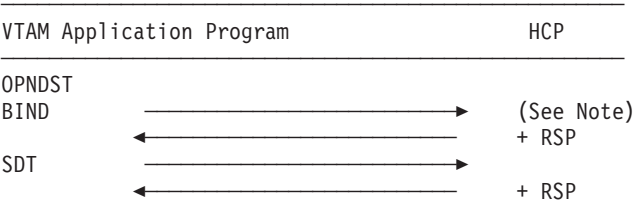
Host Communications using SDLC/SNA Communications Protocols

This section contains examples of protocol flows that a host program can use to communicate with the HCP.

Note: The examples that follow do not reflect the only way to communicate with the HCP.

Open Series

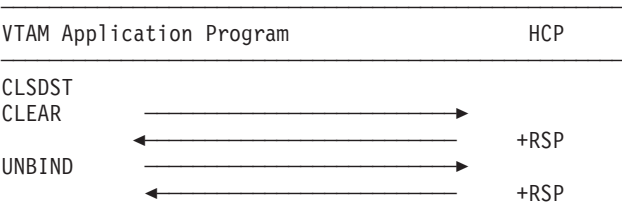
The open series is used to establish a session with the HCP LU.



Note: Bytes 4 and 5 of the BIND RU define the chaining rules followed by both HCP and the host program.

Close Series

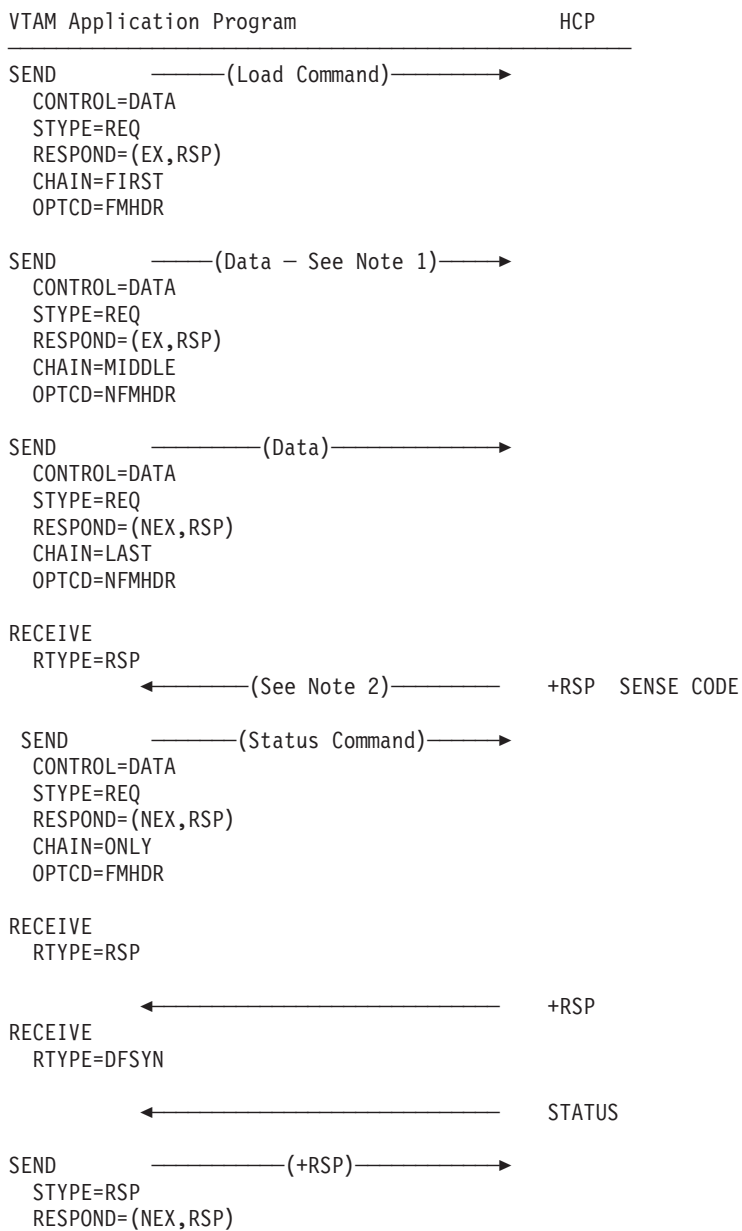
The purpose of the close series is to cause the end of a session. It is used when a transmission is complete and no more work is to be done, or when a permanent error is detected.



Load Series

Load series is used to transfer data (Toshiba modules, application, or customer code).

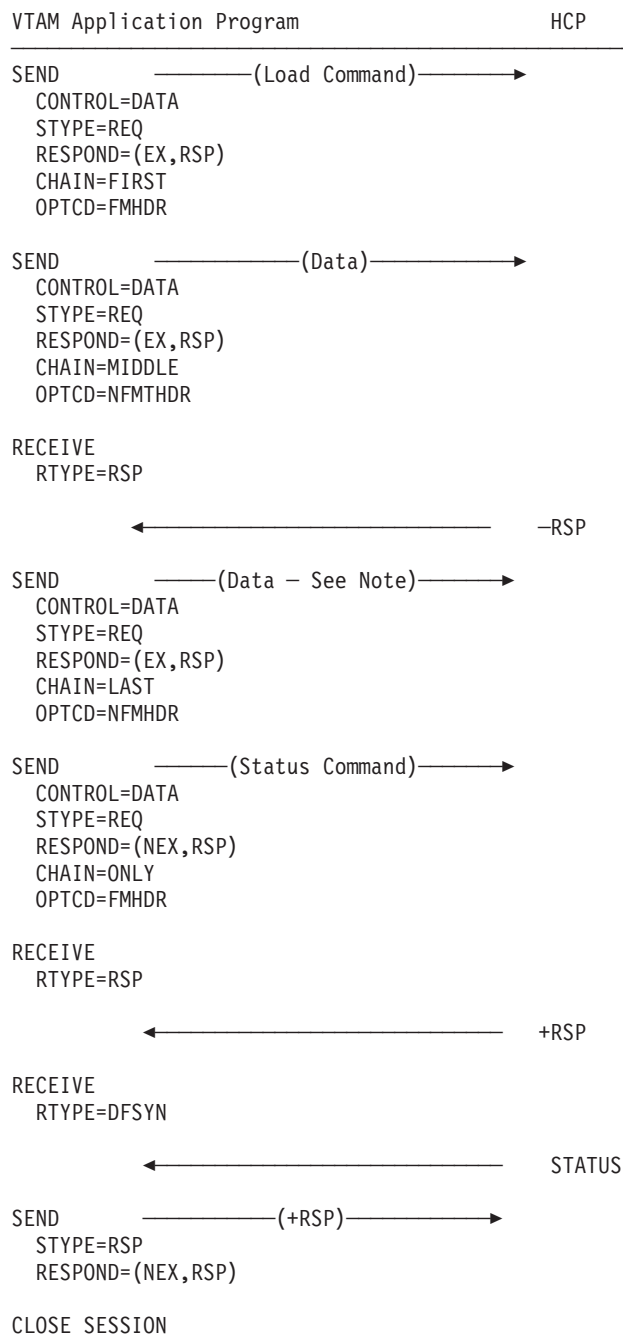
Load Series Complete With No Detected Errors



Notes:

1. This middle of chain SEND is repeated until the block before the last block of the module, file, or other unit is sent.
2. The above sequence is repeated for every module, file, or other unit contained in the LOAD command. The status is sent only after the last module.

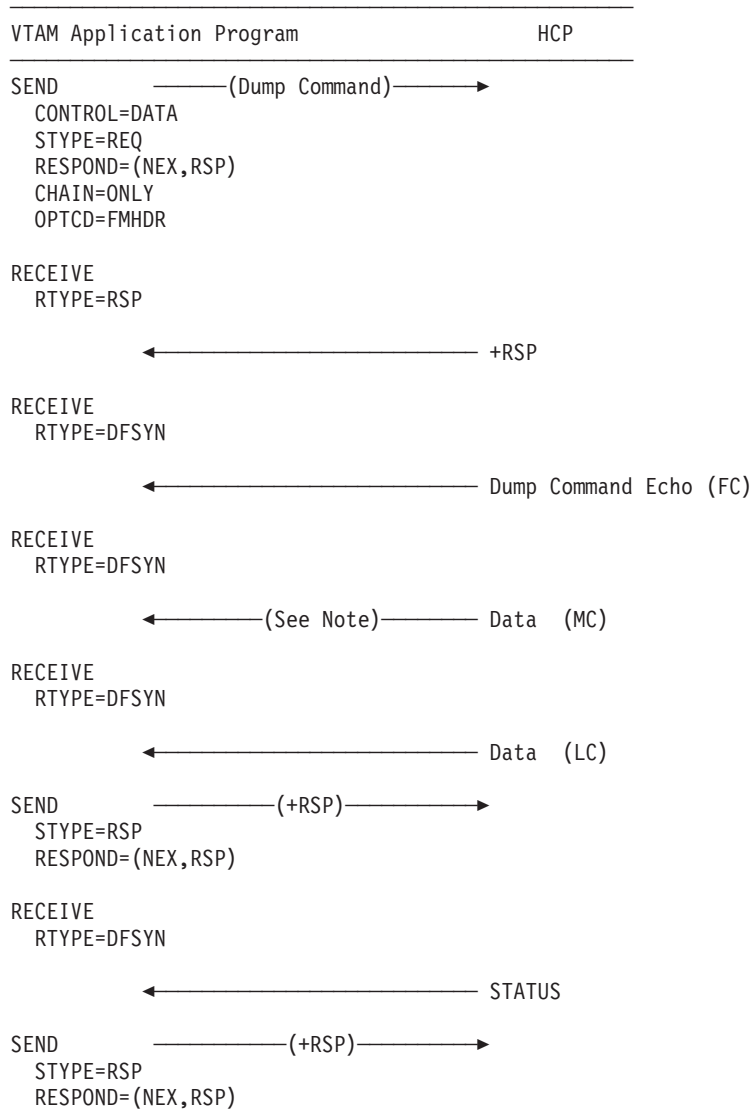
Load Series Complete With Store Controller Detected Error



Note: The data in this block has a length of 0. This is not sent if LC has already been sent.

Dump Series

The purpose of the dump series is to transmit a file from the store controller to the host. The dump series with a store controller-detected error is the same as the dump series complete, except that the status returned has the error bit set on, and an error code in the error byte. The following series is repetitive until last of chain.



Action Series

An action series command is sent to request the store controller to perform a certain function. The types of actions that can be requested are:

- Action with no data following
- Action with data following

Action Series with No Data

VTAM Application Program	HCP
--------------------------	-----

SEND —(Action Command – Create)—→
CONTROL=DATA
STYPE=REQ
RESPOND=(NEX,RSP)
CHAIN=ONLY
OPTCD=FMHDR

RECEIVE
RTYPE=RSP

← +RSP

SEND —(Status Command)—→
CONTROL=DATA
STYPE=REQ
RESPOND=(NEX,RSP)
CHAIN=ONLY
OPTCD=FMHDR

RECEIVE
RTYPE=DFSYN

← STATUS

SEND —(+RSP)—→
STYPE=RSP
RESPOND=(NEX,RSP)

Action Series with Data Following

VTAM Application Program

HCP

```

SEND      —(Action Command - See Note 1)—→
CONTROL=DATA
STYPE=REQ
RESPOND=(EX,RSP)
CHAIN=FIRST
OPTCD=FMHDR

```

SEND —————(Data - See Note 2)—————>

CONTROL=DATA
STYPE=REQ
RESPOND=(EX,RSP)
CHAIN=MIDDLE
OPTCD=NEMHDR

```

SEND      _____(Data)_____>
CONTROL=DATA
STYPE=REQ
RESPOND=(NEX,RSP)
CHAIN=LAST
OPTCD=NEMHDR

```

RECEIVE
RTYPE=RSP ← +RSP

SEND ——— (Status Command) —————>

CONTROL=DATA
STYPE=REQ
RESPOND=(NEX,RSP)
CHAIN=ONLY
OPTCD=FMHDR

RECEIVE
RTYPE=RSP ← +RSP

```

RECEIVE
  RTYPE=DFSYN
← STATUS

```

SEND $\xrightarrow{(+RSP)}$

STYPE=RSP

RESPOND=(NEX, RSP)

Notes:

1. Add, delete, read, or replace keyed records.
2. Repetitive until last of chain. Read keyed record has only one chained data element.

Appendix C. Examples of X.25 API Programs and Transaction Programs

This appendix contains examples of X.25 API programs and transaction programs (TPs). You can use these examples as guides for writing your own programs. For configuration examples for X.25 API programs and TPs, see the *4690 OS: Planning, Installation, and Configuration Guide*.

Note: You may not have all of the library functions called in these programs. Use these programs as examples only.

X.25 API Program Examples and Include Files

This section provides program examples for the X.25 API using 4680 BASIC and C. For more information on how to design these programs, see Chapter 12, “Designing an X.25 Application,” on page 229 and Chapter 13, “X.25 API Verb Reference,” on page 235. For communications configurations for sample X.25 API programs, see the *4690 OS: Planning, Installation, and Configuration Guide*.

This section contains four program examples:

XNFBRCV	This BASIC program is used to receive a file from a remote location.
XNFBSDND	This BASIC program is used to send a file to a remote location.
XNFCRCV	This C program is used to receive a file from a remote location.
XNFCSND	This C program is used to send a file to a remote location.

After describing these programs, information is provided about compiling and linking them. In addition, instructions are given for starting the requester and server programs.

XNFBRCV.B86 BASIC File Transfer Receive Program for the X.25 API

This BASIC program is used to receive a file from a remote location. The first data packet contains the file information to receive. The rest of the packets contain file data.

To start XNFBRCV as a foreground program, type the mandatory parameter “/f” at the command line.

Include Files

The following include files are included in this program code:

```
%include adxhs1f.j86      !* X.25 verb return code declaration file    */
%include adxhs5kf.j86     !* X.25 verb function prototypes      */
%include adxhs5mf.j86     !* X.25 verb return code initialization file */
```

Internal Function Definition

Function **hex\$** defined in this program translates 4-byte integer into hex string.

Internal Function Definition Code

```
!
! Translate 4-byte integer into hex string
!
function hex$(hx) public
  string hex$,errfx$
  integer*4 hx
  integer*2 s,the.sum
  integer*1 a
  errfx$ = ""
! transform error code into hexadecimal string for display
  for s = 28 to 0 step -4
    a = SHIFT(hx,s) and 000FH
    if a>9 then \
      a = a + 7
```

XNFBRCV BASIC for X.25 API

```
    errfx$ = errfx$ + chr$(a + 48)
  next S
  hex$ = errfx$
end function
```

Variables Used by External Functions

Table 77. Variables Used by External Functions

Function	Variable	Description	Type
xclose	xclose.connection.id	virtual circuit number	I*4
	xclose.facilities.field.length	length of the facility field	I*2
	xclose.facilities.field	facility field	String
	xclose.clear.user.data.length	length of the CLEAR user data	I*2
	xclose.clear.user.data	user data sent with the CLEAR	String
	xclose.cause.code	value of the cause byte of the CLEAR	String
	xclose.diagnostic.code	value of the diag. byte of the CLEAR	String
	xclose.return.code	return code	I*4
xload	xload.line.name	name of X.25 line configuration	String
	xload.line.name.length	length of the line name parameter	I*2
	xload.return.code	return code	I*4
xopenacc	xopenacc.connection.id	virtual circuit number	I*4
	xopenacc.facilities.field.length	length of the facility field	I*2
	xopenacc.facilities.field	facility field	String
	xopenacc.call.user.data.length	length of the user data	I*2
	xopenacc.call.user.data	user data	String
	xopenacc.D.bit.services	delivery confirmation: 0 = no 1 = yes	I*2
	xopenacc.return.code	return code	I*4

Table 77. Variables Used by External Functions (continued)

Function	Variable	Description	Type
xopenrec	xopenrec.first.byte.of.user.data	value of the 1st byte of user data	I*2
	xopenrec.first.byte.of.user.data.value	validate user data: 0 = no 1 = yes	String
	xopenrec.dbit.services	delivery confirmation requested: 0 = no 1 = yes	I*2
	xopenrec.calling.address.length	length of the calling PSDN address	I*2
	xopenrec.calling.address	PSDN address of the calling DTE	String
	xopenrec.called.address.length	length of the called PSDN address	I*2
	xopenrec.called.address	PSDN address of the called DTE	String
	xopenrec.call.receipt.delay	delay to wait for an incoming CALL: -1 = indefinite	I*2
	xopenrec.facilities.field.length	length of the facility field	I*2
	xopenrec.facilities.field	facility field	String
	xopenrec.call.user.data.length	length of the user data	I*2
	xopenrec.call.user.data	user data	String
	xopenrec.connection.id	virtual circuit number	I*4
	xopenrec.return.code	return code	I*4
xreceive	xreceive.connection.id	virtual circuit number	I*4
	xreceive.data.buffer.size	size of buffer	I*2
	xreceive.call.receipt.delay	data reception delay: -1 = infinite	I*2
	xreceive.data.length	length of the received data	I*2
	xreceive.data	data received	String
	xreceive.Q.bit.indicator	qualifier: 0 = normal data 1 = control data	I*2
	xreceive.M.bit.indicator	multiple: 0 = no 1 = yes	I*2
	xreceive.return.code	return code	I*4

Variable Definitions

The following variables are defined in this program:

Table 78. String Variables Used in This Program

Variable	Description	Initial value
commtail\$	command tail used	
errcode\$	error code (err)	
frmstr\$	format string	
rcvfile	receive file name	xrcvfil

XNFBRCV BASIC for X.25 API

Table 78. String Variables Used in This Program (continued)

Variable	Description	Initial value
sendfile	send file name	xsendfil
xclose.cause.code	value of the cause byte of the CLEAR	Nulls. Length = max.cause.code.len
xclose.clear.user.data	user data sent with the CLEAR	Nulls. Length = max.clear.user.data.len
xclose.diagnostic.code	value of the diag. byte of the CLEAR	Nulls. Length = max.diag.code.len
xclose.facilities.field	facility field	Nulls. Length = max.fac.field.len
xevent.cause.code	not used	Nulls. Length = max.cause.code.len
xevent.data	not used	Nulls. Length = max.data.len
xevent.diagnostic.code	not used	Nulls. Length = max.diag.code.len
xevent.facilities.field	not used	Nulls. Length = max.fac.field.len
xload.line.name	name of X.25 line configuration	X25LINE1
xopenacc.call.user.data	user data	Nulls. Length = max.call.user.data.len
xopenacc.facilities.field	facility field	Nulls. Length = max.fac.field.len
xopenrec.call.user.data	user data	Nulls. Length = max.call.user.data.len
xopenrec.called.address	PSDN address of the called DTE	Nulls. Length = max.called.addr.len
xopenrec.calling.address	PSDN address of the calling DTE	Nulls. Length = max.calling.addr.len
xopenrec.facilities.field	facility field	Nulls. Length = max.fac.field.len
xopenrec.first.byte.of.user.data.value	validate user data: 0 = no 1 = yes	Nulls. Length = max.first.byte.len
xreceive.data	data received	Nulls. Length = max.data

Table 79. Integer*2 Variables Used in This Program

Variable	Description	Initial value
background	type of application indicator	
dest.file	destination file session number	1
errnum	Errf%	
ffirst	first packet indicator	
header.data.length	header packet data length	130
max.call.user.data.len	xopenrec call user data length	128
max.called.addr.len	xopenrec called address length	17
max.calling.addr.len	xopenrec calling address length	17
max.cause.code.len	xclose cause code length	1
max.clear.user.data.len	xclose clear user data length	128
max.data	xreceive data length	32767

Table 79. Integer*2 Variables Used in This Program (continued)

Variable	Description	Initial value
max.data.len	xevent data length	1024
max.diag.code.len	xclose diagnostic code length	1
max.fac.field.len	xclose/xevent facilities field length	109
max.first.byte.len	xopenrec first byte of user length	1
max.line.name.len	xload line name length	8
rcv.buff.size	xreceive data buffer size	1024
work1i2	work variable	
work2i2	work variable	
xclose.clear.user.data.length	length of the CLEAR user data	0
xclose.facilities.field.length	length of the facility field	0
xevent.data.length	length of the xevent user data	
xevent.event.type	event type	
xevent.facilities.field.length	length of the event facilities field	
xload.line.name.length	length of the line name parameter	8
xopenacc.call.user.data.length	length of the user data	0
xopenacc.D.bit.services	delivery confirmation: 0 = no 1 = yes	0
xopenacc.facilities.field.length	length of the facility field	0
xopenrec.call.receipt.delay	delay to wait for an incoming CALL: -1 = indefinite	-1
xopenrec.call.user.data.length	length of the user data	0
xopenrec.called.address.length	length of the called PSDN address	0
xopenrec.calling.address.length	length of the calling PSDN address	0
xopenrec.dbit.services	delivery confirmation requested: 0 = no 1 = yes	0
xopenrec.facilities.field.length	length of the facility field	0
xopenrec.first.byte.of.user.data	value of the 1st byte of user data	0
xreceive.M.bit.indicator	multiple: 0 = no 1 = yes	0
xreceive.Q.bit.indicator	qualifier: 0 = normal data 1 = control data	0
xreceive.call.receipt.delay	data reception delay: -1 = indefinite	-1
xreceive.data.buffer.size	size of buffer	rcv.buff.size
xreceive.data.length	length of the received data	

XNFBRCV BASIC for X.25 API

Table 80. Integer*4 Variables Used in This Program

Variable	Description	Initial value
totalexpected	size of the file to receive	1
totalrcvd	number of bytes received	0
xclose.connection.id	virtual circuit number	xopenrec.connection.id
xclose.return.code		xclose return code
XA.OK		
xevent.connection.id	not used	xopenrec.connection.id
xevent.return.code	not used	
xload.return.code	xload return code	XA.OK
xopenacc.connection.id	virtual circuit number	xopenrec.connection.id
xopenacc.return.code	xopenacc return code	
xopenrec.connection.id	virtual circuit number	XA.OK
xopenrec.return.code	xopenrec return code	
xreceive.connection.id	virtual circuit number	
xreceive.return.code		xreceive return code
XA.OK		
xsto.return.code	xsto return code	
xsto.verb.timeout.seconds	not used	

Variable Definition Code

```

!
! Length of the fields
!
integer*2 max.fac.field.len      ! 109
integer*2 max.call.user.data.len ! 128
integer*2 max.cause.code.len     ! 1
integer*2 max.diag.code.len      ! 1

integer*2 \
max.line.name.len,      \ 8
max.called.addr.len,    \ 17
max.clear.user.data.len, \ 128
max.first.byte.len,     \ 1
max.calling.addr.len,    \ 17
max.data,               \ 32767
max.data.len,           \ 1024
rcv.buff.size,          \ 1024
header.data.len,gth     ! 130

!
! Internal variables
!
string errcode$
integer*2 errnum
integer*2 work1i2
integer*2 work2i2
string commtail$
integer*4 totalrcvd
integer*4 totalexpected
integer*2 ffirst
integer*2 background      ! task type: 1 background 0 foreground
integer*2 dest.file       ! destinate file session 1
integer*4 totalexpected

```

```
integer*4  totalrcvd
string     sendfile
string     rcvfile
string     frmstr$
```

```
!
!  X.25 Variables
!
```

```
string     xload.line.name
integer*2  xload.line.name.length
integer*4  xload.return.code
integer*4  xsto.verb.timeout.seconds
integer*4  xsto.return.code
integer*4  xreceive.connection.id
integer*2  xreceive.data.buffer.size
integer*2  xreceive.call.receipt.delay
integer*2  xreceive.data.length
string     xreceive.data
integer*2  xreceive.Q.bit.indicator
integer*2  xreceive.M.bit.indicator
integer*4  xreceive.return.code
integer*4  xclose.connection.id
integer*2  xclose.facilities.field.length
string     xclose.facilities.field
integer*2  xclose.clear.user.data.length
string     xclose.clear.user.data
string     xclose.cause.code
string     xclose.diagnostic.code
integer*4  xclose.return.code
integer*4  xevent.connection.id
integer*2  xevent.event.type
integer*2  xevent.facilities.field.length
string     xevent.facilities.field
integer*2  xevent.data.length
string     xevent.data
string     xevent.cause.code
string     xevent.diagnostic.code
integer*4  xevent.return.code

integer*2  xopenrec.first.byte.of.user.data
string     xopenrec.first.byte.of.user.data.value
integer*2  xopenrec.dbit.services
integer*2  xopenrec.calling.address.length
string     xopenrec.calling.address
integer*2  xopenrec.called.address.length
string     xopenrec.called.address
integer*2  xopenrec.call.receipt.delay
integer*2  xopenrec.facilities.field.length
string     xopenrec.facilities.field
integer*2  xopenrec.call.user.data.length
string     xopenrec.call.user.data
integer*4  xopenrec.connection.id
integer*4  xopenrec.return.code
integer*4  xopenacc.connection.id
integer*2  xopenacc.facilities.field.length
string     xopenacc.facilities.field
integer*2  xopenacc.call.user.data.length
string     xopenacc.call.user.data
integer*2  xopenacc.D.bit.services
integer*4  xopenacc.return.code
```

Variable Initialization Code

```
max.fac.field.len      = 109
max.call.user.data.len = 128
max.cause.code.len     = 1
max.diag.code.len      = 1
max.line.name.len      = 8
```

XNFBRCV BASIC for X.25 API

```
max.called.addr.len      = 17
max.clear.user.data.len  = 128
max.first.byte.len       = 1
max.calling.addr.len     = 17
max.data                 = 32767
max.data.len             = 1024
rcv.buff.size            = 1024
header.data.length       = 130
xload.line.name          = string$(max.line.name.len      , chr$(0))
xreceive.data            = string$(max.data               , chr$(0))
xclose.facilities.field  = string$(max.fac.field.len      , chr$(0))
xclose.clear.user.data   = string$(max.clear.user.data.len, chr$(0))
xclose.cause.code        = string$(max.cause.code.len     , chr$(0))
xclose.diagnostic.code   = string$(max.diag.code.len     , chr$(0))
xevent.facilities.field  = string$(max.fac.field.len      , chr$(0))
xevent.data              = string$(max.data.len           , chr$(0))
xevent.cause.code        = string$(max.cause.code.len     , chr$(0))
xevent.diagnostic.code   = string$(max.diag.code.len     , chr$(0))
xopenrec.first.byte.of.user.data.value = string$(max.first.byte.len,chr$(0))
xopenrec.calling.address  = string$(max.calling.addr.len  , chr$(0))
xopenrec.called.address  = string$(max.called.addr.len   , chr$(0))
xopenrec.facilities.field = string$(max.fac.field.len    , chr$(0))
xopenrec.call.user.data   = string$(max.call.user.data.len, chr$(0))
xopenacc.facilities.field = string$(max.fac.field.len    , chr$(0))
xopenacc.call.user.data   = string$(max.call.user.data.len, chr$(0))
```

Main Line Program

These steps show you how to run the “Main Line Code” on page 301:

1. Activate On Error routine.
2. Get the parameters passed from foreground task.
3. Determine execution mode.
Argument when the program is started as foreground must be:
 - First character - or /
 - Second character f or F
4. Set all the values control block.
5. Subroutine used to load X.25 drivers.
6. Receive an incoming call.
7. Accept an incoming call.
8. Loop round receiving data until a Clear Indication is received.
9. Begin loop.
10. Receive data on a virtual circuit.

The first data packet contains the file information. The rest of the packets contain file data.

11. If first message:
 - Get and display file name and size of the receiving file.
 - Create destination file.
12. Rest of the messages:
 - Write received data to destination file
13. End loop.
14. Close the file.
15. Disconnect the virtual circuit.
16. End of the program.

Main Line Code

```

!
! Main XNFBRCV Begins Here
! Initialization
!

on error goto mainerror
commtail$ = command$
background = 0
dest.file = 1
sendfile = "xsendfil"
rcvfile = "xrcvfil"
xopenrec.first.byte.of.user.data = 0
xopenrec.first.byte.of.user.data.value = ""

!
! Determine execution mode
!

IF LEFT$(commtail$,8) = "BACKGRND" THEN begin
    background = -1
    message$="program XNFBRCV started"
    gosub disp
    goto follow
Endif

if (len(commtail$) = 1) then begin
    xopenrec.first.byte.of.user.data = 1
    xopenrec.first.byte.of.user.data.value = commtail$
    print "Match byte = "; xopenrec.first.byte.of.user.data.value
endif else begin
    if ( (len(commtail$) <> 2) or
        ((commtail$ <> "/"f") and (commtail$ <> "/"F") and
         (commtail$ <> "-f") and (commtail$ <> "-F")) ) then begin
        print "Command line argument error - must be one character or '/f'."
        stop
    endif
endif
follow:

!
! We can now try to initialize the X.25 API.
! First set all the values control block and then call the X.25 function
!

if (not background) then print "Initializing the X.25 API"
xload.line.name = "X25LINE1"
xload.line.name.length = 8
xload.return.code = XA.OK

!
! XLOAD : Subroutine used to load X.25 drivers
!

call xload    ( xload.line.name,
                xload.line.name.length,
                xload.return.code)

!
! Display message after Xload
! If Load fails stop
!

if (xload.return.code <> XA.OK) then begin
    message$ = "Couldn't initialize X.25 API; return code = " + hex$(xload.return.code)
    gosub disp
    stop
endif else begin
    message$ = "X.25 API XLOAD done"
    gosub disp
endif

```

XNFBRCV BASIC for X.25 API

```
!
! Now we prepare to receive incoming calls
!
xopenrec.dbit.services           = 0
xopenrec.calling.address.length  = 0
xopenrec.calling.address         = ""
xopenrec.called.address.length   = 0
xopenrec.called.address          = ""
xopenrec.call.receipt.delay      = -1
xopenrec.facilities.field.length = 0
xopenrec.call.user.data.length   = 0
xopenrec.connection.id           = 0
xopenrec.return.code             = XA.OK

!
! XOPENREC : Subroutine used to receive an incoming call
!
call xopenrec ( xopenrec.first.byte.of.user.data, \
                xopenrec.first.byte.of.user.data.value, \
                xopenrec.dbit.services, \
                xopenrec.calling.address.length, \
                xopenrec.calling.address, \
                xopenrec.called.address.length, \
                xopenrec.called.address, \
                xopenrec.call.receipt.delay, \
                xopenrec.facilities.field.length, \
                xopenrec.facilities.field, \
                xopenrec.call.user.data.length, \
                xopenrec.call.user.data, \
                xopenrec.connection.id, \
                xopenrec.return.code )

!
! Display message after XOPENREC
! If XOPENREC fails stop
!
if (xopenrec.return.code <> XA.OK) then begin
    message$ = "XOPENREC failed; return code = "+ hex$(xopenrec.return.code)
    gosub disp
    stop
endif else begin
    message$ ="About to issue a call accept (XOPENACC)"
    gosub disp
endif

!
! Initialization connection.id variable
!
xopenacc.connection.id = xopenrec.connection.id
xreceive.connection.id = xopenrec.connection.id
xclose.connection.id   = xopenrec.connection.id

!
! XOPENACC : Subroutine used to accept an incoming call
! We now have an incoming call, so accept it
!
xopenacc.facilities.field.length = 0
xopenacc.facilities.field        = ""
xopenacc.call.user.data.length   = 0
xopenacc.call.user.data          = ""
xopenacc.D.bit.services          = 0
call xopenacc ( xopenacc.connection.id, \
                xopenacc.facilities.field.length, \
                xopenacc.facilities.field, \
                xopenacc.call.user.data.length, \
                xopenacc.call.user.data, \
                xopenacc.D.bit.services, \
                xopenacc.return.code )
```

```

!
!   Display message after XOPENACC
!   If XOPENACC fails stop
!
if (xopenacc.return.code <> XA.OK) then begin
    message$ = "Call Accept failed; return code = "+ hex$(xopenacc.return.code)
    gosub disp
    goto closeit
endif else begin
    message$ = "Call Accepted OK"
    gosub disp
endif

!
!   Now loop round receiving data until a Clear Indication is received.
!

ffirst = -1
totalrcvd = 0
totalexpected = 1
xreceive.return.code = XA.OK
while ((xreceive.return.code = XA.OK) and (totalrcvd < totalexpected))
    xreceive.data.buffer.size = rcv.buff.size
    xreceive.call.receipt.delay = -1
    xreceive.Q.bit.indicator = 0
    xreceive.M.bit.indicator = 0
    xreceive.return.code = XA.OK

!
!   XRECEIVE : Subroutine used to receive data on a virtual circuit
!

call xreceive( xreceive.connection.id, \
               xreceive.data.buffer.size, \
               xreceive.call.receipt.delay, \
               xreceive.data.length, \
               xreceive.data, \
               xreceive.Q.bit.indicator, \
               xreceive.M.bit.indicator, \
               xreceive.return.code )

!
!   Display message after xreceive
!   If receive fails close
!

if (xreceive.return.code <> XA.OK) then begin
    message$ = "Data Receive failed; return code = "+ hex$(xreceive.return.code)
    gosub disp
    goto closeit
endif

!
!   If this is the first data packet it contains the file information
!

if (ffirst) then begin

!
!   If wrong data length received display error message
!

    if (xreceive.data.length <> header.data.length) then begin
        message$ = "Invalid header data"
        gosub disp
        goto closeit
    endif
endif

```

XNFBRCV BASIC for X.25 API

```
!
! Message Structure
! The 10 last bytes contains the size of the file to receive
! The send file name is in position 1. The last character is blank
! Otherwise the length is 60
! The receive file name is in position 61. The last character is
! blank. Otherwise the length is 60
!

totalexpected = val(right$(xreceive.data, 10))
workli2 = match(" ", xreceive.data, 1)
if (workli2 = 0) then begin
    sendfile = left$(xreceive.data, 60)
    rcvfile = mid$(xreceive.data, 61, 60)
endif else begin
    sendfile = left$(xreceive.data, workli2-1)
    workli2 = match(" ", xreceive.data, 61)
    if (workli2 = 0) then begin
        rcvfile = mid$(xreceive.data, 61, 60)
    endif else begin
        rcvfile = mid$(xreceive.data, 61, workli2-61)
    endif
endif

!
! Display files names and size of the receiving file
!

message$ = "Receiving " + str$(totalexpected) + " bytes from "
+ sendfile + " to " + rcvfile + "."
gosub disp

!
! Create destination file
!

errcode$ = ""
CREATE rcvfile AS dest.file
if ((errcode$ <> "") and (errnum = dest.file)) then begin
    goto closeit
endif
fFirst = 0
totalrcvd = 0
endif else begin

!
! Rest of the packets
!

workli2 = len(xreceive.data)

!
! Print the length of the message received
!

if (xreceive.data.length <> workli2) then begin
    message$ = "xreceive.data.length = " + str$(xreceive.data.length) + \
        "; strlen(xreceive.data) = " + str$(workli2)
    gosub disp
    if ((xreceive.data.length < workli2) and \
        (xreceive.data.length > 0)) then begin
        xreceive.data = left$(xreceive.data, xreceive.data.length)
    endif
endif

!
! Write received data to destination file
!

frmstr$ = "C" + str$(xreceive.data.length)
WRITE FORM frmstr$; #dest.file; xreceive.data
WRITE #dest.file; xreceive.data
totalrcvd = totalrcvd + xreceive.data.length
```

```

        message$ = "Received "+ str$(totalrcvd)+ "bytes of "+ str$(totalexpected)
        gosub disp
    endif
wend

!
!   Close the file
!

CLOSE dest.file
message$ = "File Transfer Complete"
gosub disp

!
!   Display message after receiving the file
!

if (xreceive.return.code <> XA.OK) then begin
    message$ = "Unexpected type of data has been received"
    gosub disp
endif else begin
    message$ = "End of data Indicator received"
    gosub disp
endif
endif

closeit:
!
!   Closing circuit
!

message$ = "Closing circuit..."
gosub disp

!
!   Now that all data has been sent the call is cleared
!

xclose.facilities.field.length = 0
xclose.facilities.field        = ""
xclose.clear.user.data.length = 0
xclose.clear.user.data         = ""
xclose.cause.code              = "C"
xclose.diagnostic.code         = "N"
xclose.return.code = XA.OK

!
!   XCLOSE : Subroutine used to disconnect a switched virtual circuit
!

call xclose( xclose.connection.id,          \
             xclose.facilities.field.length, \
             xclose.facilities.field,        \
             xclose.clear.user.data.length, \
             xclose.clear.user.data,         \
             xclose.cause.code,              \
             xclose.diagnostic.code,         \
             xclose.return.code )

!
!   Display message after xclose
!   If close fails stop
!

if (xclose.return.code <> XA.OK) then begin
    message$ = "XCLOSE failed; return code = "+ hex$(xclose.return.code)
    gosub disp
    stop
endif else begin
    message$ = "Call Cleared OK"
    gosub disp
endif
message$ = "Program finished"
gosub disp
stop

```

XNFBRCV BASIC for X.25 API

```

!
!
!
Display subroutine

disp:
  if (not background) then begin
    print Message$
  endif Else Begin
    CALL ADXSERVE(RET,26,LEN(MESSAGE$),MESSAGE$)
    Wait ;2000
  endif

return

!
!
!
End of the program

stop

```

On Error Routine

This subroutine prints error information if a foreground program. If the error code is RN, this subroutine ends the program. Otherwise, the program resumes.

On Error Routine Code

```
mainerror:
  errcode$ = err
  errnum   = errf%
  if (not background) then begin
    print "errcode$ = "; errcode$
    print "errnum = "; str$(errnum)
  endif
  if (errcode$ <> "RN") then begin
    resume
  endif
  stop
end
```

XNFBSD.B86 BASIC File Transfer Send Program for the X.25 API

This BASIC program is used to send a file to a remote location. The input parameters are:

- Virtual circuit name
- File to send
- File name to copy to
- Remote application name if this program starts it (0 if you do accept the default name)
- Call match byte

Include Files

The following include files are included in this program code:

```
%include adxhs5lf.j86      !* X.25 verb return code declaration file
%include adxhs5kf.j86      !* X.25 verb function prototypes
%include adxhs5mf.j86      !* X.25 verb return code initialization file
```

Internal Function Definition

Function **hex\$** defined in this program translates a 4-byte integer into a hex string.

Internal Function Definition Code

```
function hex$(hx) public
  string hex$,errfx$
  integer*4 hx
  integer*2 s,the.sum
```

```

integer*1 a
errfx$ = ""
for s = 28 to 0 step -4
  a = SHIFT(hx,s) and 000FH
  if a>9 then \
    a = a + 7
  errfx$ = errfx$ + chr$(a + 48)
next S
hex$ = errfx$
end function

```

Variables Used by External Functions

Table 81. Variables Used by External Functions

Function	Variable	Description	Type
xclose	xclose.connection.id	virtual circuit number	I*4
	xclose.facilities.field.length	length of the facility field	I*2
	xclose.facilities.field	facility field	String
	xclose.clear.user.data.length	length of the CLEAR user data	I*2
	xclose.clear.user.data	user data send with the CLEAR	String
	xclose.cause.code	value of the cause byte of the CLEAR	String
	xclose.diagnostic.code	value of the diag. byte of the CLEAR	String
	xclose.return.code	return code	I*4
xevent	xevent.cause.code	value of the cause byte received with the clear or reset	String
	xevent.connection.id	SVC number	I*4
	xevent.data	data received with the clear or reset	String
	xevent.data.length	length of data received	I*2
	xevent.diagnostic.code	value of the diag. byte received with the clear or reset	String
	xevent.event.type	type of event detected	I*2
	xevent.facilities.field	facility field	String
	xevent.facilities.field.length	length of the facility field	I*2
	xevent.return.code	return code	I*4
xload	xload.line.name	name of X.25 line configuration	String
	xload.line.name.length	length of the line name parameter	I*2
	xload.return.code	return code	I*4

XNFBSD BASIC for X.25 API

Table 81. Variables Used by External Functions (continued)

Function	Variable	Description	Type
xopen	xopen.D.bit.services	delivery confirmation: 0 = no 1 = yes	I*2
	xopen.call.user.data	user data	String
	xopen.call.user.data.length	length of user data	I*2
	xopen.called.address	PSDN address of the called dte	String
	xopen.called.address.length	length of the called PSDN address	I*2
	xopen.xopen.connection.id	SVC number	I*4
	xopen.facilities.field	facility field	String
	xopen.facilities.field.length	length of the facility field	I*2
	xopen.remote.application	name of remote 4680 or 4690 application	String
	xopen.remote.application.length	length of the remote application	I*2
	xopen.return.code	return code	I*4
	xopen.vcr.name	name of virtual circuit record	String
	xopen.vcr.name.length	length of the user data	I*2
xsend	xsend.D.bit.indicator	delivery confirmation: 0 = no 1 = yes	I*2
	xsend.M.bit.indicator	multiple: 0 = no 1 = yes	I*2
	xsend.Q.bit.indicator	qualifier: 0 = normal data 1 = control data	I*2
	xsend.connection.id	SVC number	I*4
	xsend.data	data to be sent	String
	xsend.data.bytes.sent	length of the data already sent	I*2
	xsend.data.length	length of the data buffer	I*2
	xsend.return.code	return code	I*4

Table 81. Variables Used by External Functions (continued)

Function	Variable	Description	Type
receive	xreceive.connection.id	virtual circuit number	I*4
	xreceive.data.buffer.size	size of buffer	I*2
	xreceive.call.receipt.delay	data reception delay: -1 = infinite	I*2
	xreceive.data.length	length of the received data	I*2
	xreceive.data	data received	String
	xreceive.Q.bit.indicator	qualifier: 0 = normal data 1 = control data	I*2
	xreceive.M.bit.indicator	multiple: 0 = no 1 = yes	I*2
	xreceive.return.code	return code	I*4

Variable Definitions

The following variables are defined in this program:

Table 82. String Variables Used in This Program

Variable	Description	Initial value
arg\$	1st string argument	
commtail\$	command tail used	
frmstr\$	format string	
rcvfile	receive file name	xrcvfil
sendfile	send file name	xsendfil
xclose.cause.code	value of the cause byte of the CLEAR	Nulls. Length = max.cause.code.len
xclose.clear.user.data	user data send with the CLEAR	Nulls. Length = max.clear.user.data.len
xclose.diagnostic.code	value of the diag. byte of the CLEAR	Nulls. Length = max.diag.code.len
xclose.facilities.field	facility field	Nulls. Length = max.fac.field.len
xevent.cause.code	value of the cause byte received with the clear or reset	Nulls. Length = max.cause.code.len
xevent.data	data received with the clear or reset	Nulls. Length = max.event.data.len
xevent.diagnostic.code	value of the diag. byte received with the clear or reset	Nulls. Length = max.diag.code.len
xevent.facilities.field	facility field	Nulls. Length = max.fac.field.len
xload.line.name	name of X.25 line configuration	X25LINE1
xopen.call.user.data	user data	Nulls. Length = max.call.user.data.len
xopen.called.address	PSDN address of the called DTE	Nulls. Length = max.called.addr.len
xopen.facilities.field	facility field	Nulls. Length = max.fac.field.len
xopen.remote.application	name of remote 4680 or 4690 application	Nulls. Length = max.remote.appl.len
xopen.vcr.name	name of virtual circuit record	Nulls. Length = max.vcr.name.len
xsend.data	data to be sent	Nulls. Length = max.data
xreceive.data	data received	Nulls. Length = max.data

Table 83. Integer*2 Variables Used in This Program

Variable	Description	Initial value
arg.cnt	argument number	0
arg.del	argument initial position	1
arg.len	argument length	
event.clear	clear event	1
filesize	number of bytes sent	size(sendfile)
send.buff.size	send buffer size	1024
src.file	session number for the send file	1
start.arg	variable used to parse the arguments	
max.call.user.data.len	XOPEN call user data length	128
max.called.addr.len	XOPEN called address length	17
max.cause.code.len	XCLOSE/XEVENT cause code length	1
max.clear.user.data.len	XCLOSE clear user data length	128
max.data	XSEND/SRECEIVE data length	32767
max.diag.code.len	XCLOSE/XEVENT diagnostic code length	1
max.event.data.len	XEVENT data length	128
max.fac.field.len	XCLOSE/XEVENXOPEN facilities field length	109
max.line.name.len	XLOAD line name length	8
max.remote.appl.len	remote application length	8
max.vcr.name.len	XOPEN vcr name length	8
xclose.clear.user.data.length	length of the CLEAR user data	0
xclose.facilities.field.length	length of the facility field	0
xevent.data.length	length of the XEVENT user data	0
xevent.event.type	type of event detected	0
xevent.facilities.field.length	length of the event facilities field	0
xload.line.name.length	length of the line name parameter	8
xopen.D.bit.services	delivery confirmation: 0 = no 1 = yes	0
xopen.call.user.data.length	length of user data	0
xopen.called.address.length	length of the called PSDN address	0
xopen.facilities.field.length	length of the facility field	0
xopen.remote.application.length	length of the remote application	0
xopen.vcr.name.length	length of the user data	6
xsend.D.bit.indicator	delivery confirmation: 0 = no 1 = yes	0
xsend.M.bit.indicator	multiple: 0 = no 1 = yes	0

Table 83. Integer*2 Variables Used in This Program (continued)

Variable	Description	Initial value
xsend.Q.bit.indicator	qualifier: 0 = normal data 1 = control data	0
xsend.data.bytes.sent	length of the data already sent	0
xsend.data.length	length of the data buffer	
xreceive.M.bit.indicator	multiple: 0 = no 1 = yes	0
xreceive.Q.bit.indicator	qualifier: 0 = normal data 1 = control data	0
xreceive.call.receipt.delay	data reception delay: -1 = infinite	-1
xreceive.data.buffer.size	size of buffer	128
xreceive.data.length	length of the received data	0

Table 84. Integer*4 Variables Used in This Program

Variable	Description	Initial value
bytessent	number of bytes sent	0
xclose.connection.id	virtual circuit number	xopen.connection.id
xclose.return.code	XCLOSE return code	
xevent.connection.id	virtual circuit number	xopen.connection.id
xevent.return.code	not used	XA.OK
xload.return.code	XLOAD return code	XA.OK
xopen.connection.id	virtual circuit number	0
xopen.return.code	XOPEN return code	0
xsend.connection.id	virtual circuit number	xopen.connection.id
xsend.return.code	XSEND return code	XA.OK
xreceive.connection.id	virtual circuit number	xopen.connection.id
xreceive.return.code	XRECEIVE return code	XA.OK
xsto.return.code	XSTO return code	
xsto.verb.timeout.seconds	not used	

Variable Definition Code

```

!
!
!   Length of the fields
!
integer*2  max.fac.field.len      ! 109
integer*2  max.call.user.data.len ! 128
integer*2  max.event.data.len     ! 128
integer*2  max.cause.code.len     ! 1
integer*2  max.diag.code.len      ! 1
integer*2
max.line.name.len,               \ 8
max.vcr.name.len,                \ 8
max.remote.appl.len,             \ 8
max.called.addr.len,             \ 17

```

XNFBSND BASIC for X.25 API

```
max.clear.user.data.len, \ 128
max.data,                \ 32767
send.buff.size           \ 1024
event.clear              ! 1
```

```
!
!
!
```

X.25 Variables

```
string      xload.line.name
integer*2   xload.line.name.length
integer*4    xload.return.code
string      xopen.vcr.name
integer*2    xopen.vcr.name.length
string      xopen.remote.application
integer*2    xopen.remote.application.length
integer*2    xopen.D.bit.services
integer*2    xopen.called.address.length
string      xopen.called.address
integer*2    xopen.facilities.field.length
string      xopen.facilities.field
integer*2    xopen.call.user.data.length
string      xopen.call.user.data
integer*4    xopen.connection.id
integer*4    xopen.return.code
integer*4    xsend.connection.id
integer*2    xsend.data.length
string      xsend.data
integer*2    xsend.Q.bit.indicator
integer*2    xsend.M.bit.indicator
integer*2    xsend.D.bit.indicator
integer*2    xsend.data.bytes.sent
integer*4    xsend.return.code
integer*4    xsto.verb.timeout.seconds
integer*4    xsto.return.code
integer*4    xreceive.connection.id
integer*2    xreceive.data.buffer.size
integer*2    xreceive.call.receipt.delay
integer*2    xreceive.data.length
string      xreceive.data
integer*2    xreceive.Q.bit.indicator
integer*2    xreceive.M.bit.indicator
integer*4    xreceive.return.code
integer*4    xclose.connection.id
integer*2    xclose.facilities.field.length
string      xclose.facilities.field
integer*2    xclose.clear.user.data.length
string      xclose.clear.user.data
string      xclose.cause.code
string      xclose.diagnostic.code
integer*4    xclose.return.code
integer*4    xevent.connection.id
integer*2    xevent.event.type
integer*2    xevent.facilities.field.length
string      xevent.facilities.field
integer*2    xevent.data.length
string      xevent.data
string      xevent.cause.code
string      xevent.diagnostic.code
integer*4    xevent.return.code
```

```
!
!
!
```

Internal variables

```
string      arg$                !* 1st string argument
integer*2    arg.cnt
integer*2    arg.del
```

```

integer*2  arg.len
integer*4  bytessent      !* no of bytes sent
string     commtail$      !* command line arguments
integer*2  filesize      !* no of bytes sent
string     frmstr$
integer*2  len.commtail
string     rcvfile
string     sendfile
integer*2  src.file
integer*2  start.arg

```

Variable Initialization Code

```

max.fac.field.len      = 109
max.call.user.data.len = 128
max.event.data.len     = 128
max.cause.code.len     = 1
max.diag.code.len      = 1
max.line.name.len      = 8
max.vcr.name.len       = 8
max.remote.appl.len    = 8
max.called.addr.len    = 17
max.clear.user.data.len = 128
max.data               = 32767
max.data.len           = 1024
send.buff.size         = 1024
event.clear            = 1
xload.line.name        = string$(max.line.name.len , CHR$(0))
xclose.cause.code      = string$(max.cause.code.len , CHR$(0))
xclose.clear.user.data = string$(max.clear.user.data.len, CHR$(0))
xclose.diagnostic.code = string$(max.diag.code.len , CHR$(0))
xclose.facilities.field = string$(max.fac.field.len , CHR$(0))
xevent.cause.code      = string$(max.cause.code.len , CHR$(0))
xevent.data            = string$(max.event.data.len , CHR$(0))
xevent.diagnostic.code = string$(max.diag.code.len , CHR$(0))
xevent.facilities.field = string$(max.fac.field.len , CHR$(0))
xopen.call.user.data   = string$(max.call.user.data.len , CHR$(0))
xopen.called.address   = string$(max.called.addr.len , CHR$(0))
xopen.facilities.field = string$(max.fac.field.len , CHR$(0))
xopen.remote.application = string$(max.remote.appl.len , CHR$(0))
xopen.vcr.name         = string$(max.vcr.name.len , CHR$(0))
xreceive.data          = string$(max.data , CHR$(0))
xsend.data             = string$(max.data , CHR$(0))

```

Main Line Program

These steps show you how to run the “Main Line Code” on page 314.

1. Variable initialization.
2. Enable ON ERROR routine.
3. If command line arguments, parse the parameters:
 - Argument 1: VCR Name
 - Argument 2: Send file name
 - Argument 3: Receive file name
 - Argument 4: Remote application name
 - Argument 5: Matching criteria “first byte”
4. Open the file to send.
5. Initialize the X.25 API.
6. Make the call XOPEN.
7. Send the required number of data packets.

The packets will have the D-bit set, so the verb will be completed only after the remote application has issued an acknowledgment.

8. Wait for CLEAR packet.
9. Ensure CLEAR has been received.
10. Close this end of the circuit.

Main Line Code

```

!
! Main 4680SEND.BAS begins here
!

on error goto mainerror
src.file = 1
sendfile = "xsendfil"
rcvfile = "xrcvfil"
xopen.vcr.name = "L1SVC1"
xopen.vcr.name.length = 6
xopen.remote.application = ""
xopen.remote.application.length = 0
xopen.call.user.data = ""
xopen.call.user.data.length = 0
commtail$ = command$

!
! If command line arguments, parse the script files
!

len.commtail = len(commtail$)
if (len.commtail > 0) then begin
    arg.cnt = 0
    arg.del = 1
    start.arg = 1
    while (arg.del > 0)
        arg.cnt = arg.cnt + 1
        arg.del = match(" ", commtail$, start.arg)
        if (arg.del = 0) then begin
            arg.len = len.commtail-start.arg+1
            arg$ = right$(commtail$,arg.len)
        endif else begin
            arg.len = arg.del-start.arg
            arg$ = mid$(commtail$, start.arg, arg.len)
            start.arg = arg.del + 1
        endif
        on (arg.cnt) gosub case.arg.1, case.arg.2, case.arg.3, case.arg.4, \
            case.arg.5, case.arg.6
    goto default:
case.arg.1:
!
! Argument 1: VCR Name
!
    xopen.vcr.name = arg$
    xopen.vcr.name.length = arg.len
    return
case.arg.2:
!
! Argument 2: Send file name
!
    sendfile = arg$
    return
case.arg.3:
!
! Argument 3: Receive file name
!
    rcvfile = arg$
    return
case.arg.4:
!
! Argument 4: Remote application name
!

```

```

xopen.remote.application = arg$
xopen.remote.application.length = arg.len
return

case.arg.5:
!
!   Argument 5: Matching criteria 'first byte'
!
xopen.call.user.data = arg$
xopen.call.user.data.length = 1
return

case.arg.6:
!
!   Argument 5: Ignored if any
!
print "Max of 5 arguments allowed; others ignored."
return

default:
wend
endif

!
!   Display parameters
!

print "Source file = "; sendfile
print "Destination file = "; rcvfile
print "VCR name = "; xopen.vcr.name

!
!   Open the file to send
!

open sendfile as src.file nowrite model
filesize = size(sendfile)
print "Source file size = "; FileSize

!
!   We can now initialize the X.25 API.
!   First call an internal function to set all the values in the
!   control block and then call the X.25 function.
!

print "Initializing the X.25 API (XLOAD)"
xload.line.name = "X25LINE1"
xload.line.name.length = 8
xload.return.code = XA.OK
call xload(xload.line.name, \
          xload.line.name.length, \
          xload.return.code)
if (xload.return.code <> XA.OK) then begin
print "Couldn't initialize X.25 API; return code = "; \
hex$(xload.return.code)
stop
endif else begin
print "X.25 API XLOAD done"
endif

!
!   Now that the API is initialized, the call can be placed. Again
!   an internal function is called to set up all the control information
!   and then the API function is called. After getting the immediate
!   return code, the program waits for the associated semaphore to be
!   cleared, indicating completion of the function. The call should now
!   have been accepted by the remote application. The immediate and
!   completion return codes are checked. A special case is considered of
!   the function failing because the call was cleared.
!
```

XNFBSD BASIC for X.25 API

```
print "Making the call (XOPEN)"
xopen.D.bit.services      = 0
xopen.called.address.length = 0
xopen.facilities.field.length = 0
xopen.connection.id      = 0
xopen.return.code        = 0
call xopen(xopen.vcr.name, \
           xopen.vcr.name.length, \
           xopen.remote.application, \
           xopen.remote.application.length, \
           xopen.D.bit.services, \
           xopen.called.address.length, \
           xopen.called.address, \
           xopen.facilities.field.length, \
           xopen.facilities.field, \
           xopen.call.user.data.length, \
           xopen.call.user.data, \
           xopen.connection.id, \
           xopen.return.code) !
if (xopen.return.code <> XA.OK ) then begin
  print "Couldn't make XOPEN call; return code = "; hex$(xopen.return.code)
  stop
endif else begin
  xsend.connection.id = xopen.connection.id
  xreceive.connection.id = xopen.connection.id
  xevent.connection.id = xopen.connection.id
  xclose.connection.id = xopen.connection.id
  print "Connection established to remote DTE"
endif
!
! Now send the required number of data packets. The packets will
! have the D-bit set, so the verb will be completed only after
! the remote application has issued an acknowledgment.
!
print "Sending "; sendfile; " to "; rcvfile; \
      " on VCR circuit "; xopen.vcr.name
ffirst = -1
bytessent = 0
while (bytessent < filesize)
  if (ffirst) then begin
    xsend.data = left$(sendfile + string$(60," "), 60) + \
                 left$(rcvfile + string$(60," "), 60) + \
                 left$(str$(FileSize) + string$(10," "), 10)
    xsend.data.length = 130
  endif else begin
    if (filesize - bytessent < send.buff.size) then begin
      xsend.data.length = FileSize - BytesSent
    endif else begin
      xsend.data.length = send.buff.size
    endif
    frmstr$ = "C" + str$(xsend.data.length)
    read form frmstr$; #src.file; xsend.data
  endif
  xsend.Q.bit.indicator = 0
  xsend.M.bit.indicator = 0
  xsend.D.bit.indicator = 0
  xsend.data.bytes.sent = 0
  xsend.return.code = XA.OK
  call xsend(xsend.connection.id, \
             xsend.data.length, \
             xsend.data, \
             xsend.Q.bit.indicator, \
             xsend.M.bit.indicator, \
             xsend.D.bit.indicator, \
             xsend.data.bytes.sent, \
             xsend.return.code)
```



```

if (xsend.return.code <> XA.OK) then begin
  print "Send failed; return code = "; hex$(xsend.return.code)
  goto closeit
endif else begin
  if (fFirst) then begin
    fFirst = 0
    print "Sent header data: "; xsend.data.bytes.sent; " bytes."
  endif else begin
    BytesSent = BytesSent + xsend.data.bytes.sent
    print "Sent "; BytesSent; " bytes of "; FileSize
  endif
endif
endif
wend

print "Sent data OK. Waiting for CLEAR packet..."
close src.file

!
! Now wait for clear packet
!

xevent.event.type = 0
while (xevent.event.type <> event.clear)
  xreceive.data.buffer.size = 128
  xreceive.call.receipt.delay = -1
  xreceive.data.length = 0
  xreceive.Q.bit.indicator = 0
  xreceive.M.bit.indicator = 0
  xreceive.return.code = XA.OK
  call xreceive(xreceive.connection.id, \
    xreceive.data.buffer.size, \
    xreceive.call.receipt.delay, \
    xreceive.data.length, \
    xreceive.data, \
    xreceive.Q.bit.indicator, \
    xreceive.M.bit.indicator, \
    xreceive.return.code)
  if (xreceive.return.code <> XA.EVENT) then begin
    print "Unexpected RC from Receive = "; hex$(xreceive.return.code)
    goto closeit
  endif
  print "XEVENT..."
!
! Ensure clear has been received
!

xevent.event.type = 0
xevent.facilities.field.length = 0
xevent.data.length = 0
xevent.return.code = XA.OK
call xevent ( xevent.connection.id, \
  xevent.event.type, \
  xevent.facilities.field.length, \
  xevent.facilities.field, \
  xevent.data.length, \
  xevent.data, \
  xevent.cause.code, \
  xevent.diagnostic.code, \
  xevent.return.code )
if (xevent.return.code <> XA.OK) then begin
  print "Error receiving Event; return code = "; hex$(xevent.return.code)
  goto closeit
endif else begin
  if (xevent.event.type = EVENT.CLEAR) then begin
    print "Clear indication packet received"
  endif else begin
    print "Unexpected type of data has been received"
  endif
endif
endif

```

XNFBSD BASIC for X.25 API

```
wend

closeit:
print "XCLOSE..."

!
!   Close this end of the circuit
!

xclose.facilities.field.length = 0
xclose.facilities.field       = ""
xclose.clear.user.data.length = 0
xclose.clear.user.data        = ""
xclose.cause.code              = "C"
xclose.diagnostic.code         = "N"
call xclose( xclose.connection.id, \
             xclose.facilities.field.length, \
             xclose.facilities.field, \
             xclose.clear.user.data.length, \
             xclose.clear.user.data, \
             xclose.cause.code, \
             xclose.diagnostic.code, \
             xclose.return.code )
if (xclose.return.code <> XA.OK) then begin
  print "Clear failed; return code = "; hex$(xclose.return.code)
  stop
endif
print "Program finished successfully"
stop
```

On Error Routine

Stop if error code is RN, otherwise resume.

On Error Routine Code

```
mainerror:
  if (err <> "RN") then begin
    resume
  endif
  stop
end
```

Include Files for 4680 BASIC X.25 Application Programs

This section contains the include files to be used with BASIC X.25 application programs that call the operating system X.25 API.

X.25 Verb File: ADXHS5KF.J86

This is the X.25 verb file to be included by 4680 BASIC programs that call the X.25 API.

X.25 API verb prototypes for 4680 BASIC

XLOAD

```

sub xload ( line.name, line.name.length, return.code ) external
  STRING   line.name
  INTEGER*2 line.name.length
  INTEGER*4 return.code
end sub

```

XOPEN

```

sub xopen ( vcr.name,           \
            vcr.name.length,    \
            remote.application, \
            remote.application.length, \
            D.bit.services,     \
            called.address.length, \
            called.address,     \
            facilities.field.length, \
            facilities.field,    \
            call.user.databuff.length, \
            call.user.databuff,  \
            connection.id,      \
            return.code ) external
  STRING   vcr.name
  INTEGER*2 vcr.name.length
  STRING   remote.application
  INTEGER*2 remote.application.length
  INTEGER*2 D.bit.services
  INTEGER*2 called.address.length
  STRING   called.address
  INTEGER*2 facilities.field.length
  STRING   facilities.field
  INTEGER*2 call.user.databuff.length
  STRING   call.user.databuff
  INTEGER*4 connection.id
  INTEGER*4 return.code
end sub

```

X.25 BASIC Verb Include File

XSEND

```
sub xsend ( connection.id, \
            databuff.length, \
            databuff, \
            Q.bit.indicator, \
            M.bit.indicator, \
            D.bit.indicator, \
            data.bytes.sent, \
            return.code ) external
INTEGER*4 connection.id
INTEGER*2 databuff.length
STRING databuff
INTEGER*2 Q.bit.indicator
INTEGER*2 M.bit.indicator
INTEGER*2 D.bit.indicator
INTEGER*2 data.bytes.sent
INTEGER*4 return.code
end sub
```

XRECEIVE

```
sub xreceive ( connection.id, \
               databuff.buffer.size, \
               call.receipt.delay, \
               databuff.length, \
               databuff, \
               Q.bit.indicator, \
               M.bit.indicator, \
               return.code ) external
INTEGER*4 connection.id
INTEGER*2 databuff.buffer.size
INTEGER*2 call.receipt.delay
INTEGER*2 databuff.length
STRING databuff
INTEGER*2 Q.bit.indicator
INTEGER*2 M.bit.indicator
INTEGER*4 return.code
end sub
```

XCLOSE

```

sub  xclose  ( connection.id,          \
                facilities.field.length, \
                facilities.field,        \
                clear.user.databuff.length, \
                clear.user.databuff,      \
                cause.code,              \
                diagnostic.code,          \
                return.code ) external
INTEGER*4 connection.id
INTEGER*2 facilities.field.length
STRING   facilities.field
INTEGER*2 clear.user.databuff.length
STRING   clear.user.databuff
STRING   cause.code
STRING   diagnostic.code
INTEGER*4 return.code

```

XEVENT

```

sub  xevent  ( connection.id,          \
                event.type,            \
                facilities.field.length, \
                facilities.field,        \
                databuff.length,        \
                databuff,               \
                cause.code,             \
                diagnostic.code,         \
                return.code ) external
INTEGER*4 connection.id
INTEGER*2 event.type
INTEGER*2 facilities.field.length
STRING   facilities.field
INTEGER*2 databuff.length
STRING   databuff
STRING   cause.code
STRING   diagnostic.code
INTEGER*4 return.code
end sub

```

XSTO

```

sub  xsto ( verb.timeout.seconds, return.code ) external
INTEGER*4 verb.timeout.seconds
INTEGER*4 return.code
end sub

```

X.25 BASIC Verb Include File

XOPENREC

```
sub  xopenrec ( first.byte.of.user.databuff, \
                first.byte.of.user.databuff.value, \
                D.bit.services, \
                calling.address.length, \
                calling.address, \
                called.address.length, \
                called.address, \
                call.receipt.delay, \
                facilities.field.length, \
                facilities.field, \
                call.user.databuff.length, \
                call.user.databuff, \
                connection.id, \
                return.code ) external
INTEGER*2 first.byte.of.user.databuff
STRING    first.byte.of.user.databuff.value
INTEGER*2 D.bit.services
INTEGER*2 calling.address.length
STRING    calling.address
INTEGER*2 called.address.length
STRING    called.address
INTEGER*2 call.receipt.delay
INTEGER*2 facilities.field.length
STRING    facilities.field
INTEGER*2 call.user.databuff.length
STRING    call.user.databuff
INTEGER*4 connection.id
INTEGER*4 return.code
end sub
```

XOPENACC

```
sub  xopenacc ( connection.id, \
                facilities.field.length, \
                facilities.field, \
                call.user.databuff.length, \
                call.user.databuff, \
                D.bit.services, \
                return.code ) external
INTEGER*4 connection.id
INTEGER*2 facilities.field.length
STRING    facilities.field
INTEGER*2 call.user.databuff.length
STRING    call.user.databuff
INTEGER*2 D.bit.services
INTEGER*4 return.code
end sub
```

XIT

```

sub xit      ( connection.id,      \
               interrupt.databuff.length, \
               interrupt.databuff,    \
               return.code ) external
  INTEGER*4 connection.id
  INTEGER*2 interrupt.databuff.length
  STRING    interrupt.databuff
  INTEGER*4 return.code
end sub

```

XRESET

```

sub xreset   ( connection.id,      \
               cause.code,         \
               diagnostic.code,    \
               return.code ) external
  INTEGER*4 connection.id
  STRING    cause.code
  STRING    diagnostic.code
  INTEGER*4 return.code
end sub

```

XALLOC

```

sub xalloc   ( vcr.name, vcr.name.length, connection.id, return.code ) external
  STRING     vcr.name
  INTEGER*2  vcr.name.length
  INTEGER*4  connection.id
  INTEGER*4  return.code
end sub

```

XDEALLOC

```

sub xdealloc ( connection.id, return.code ) external
  INTEGER*4 connection.id
  INTEGER*4 return.code
end sub

```

XWAIT

```

sub XWAIT ( num.waits, wait.time, wait.ids, wait.types, wait.event ) external
  INTEGER*2 num.waits
  INTEGER*4 wait.time
  INTEGER*4 wait.ids
  INTEGER*1 wait.types
  INTEGER*4 wait.event
end sub

```

X.25 Verb Return Code Declaration File: ADXHS5LF.J86

This is the X.25 verb return code declaration file to be included by 4680 BASIC programs that call the X.25 API. It is used in conjunction with the return code initialization file ADXHS5MF.J86. Table 89 on page 347 specifies the return codes.

X.25 BASIC Initialization Include File

X.25 Verb Return Code Initialization File: ADXHS5MF.J86

This is the X.25 API verb return code initialization file, which is on the 4690 Optionals. It should be included by 4680 BASIC programs. See Table 89 on page 347 for a description of what it contains.

XNFCRCV.HIC File Transfer Receive C Program for the X.25 API

This C program is used to receive a file from a remote location. The first data packet contains the file information to receive. The rest of the packets contain file data. XNFCSDND can be used at the remote 4690.

The following operating system programs use X25LINE1 as the default line name and L1SVC1 as the default VCR name.

The input parameter is /F.

Include Files

The following include files are included in this program code:

stdio.h	Standard include files
stdlib.h	Standard include files
string.h	Standard include files
portabm.h	Standard include files
adxhs5lc.i86	X.25 API Return Codes
adxhs5kx.x86	X.25 API function prototypes

Variables Used by Verb Structures

Table 85. Verb Structure Variables

Function	Variable	Description	Type
close	connection_id	virtual circuit number	LONG
	facilities_len	length of the facility field	WORD
	facilities_data[128]	facility field	CHAR
	user_data_len	length of the CLEAR user data	WORD
	user_data[128]	user data sent with the CLEAR	CHAR
	cause	value of the cause byte of the CLEAR	CHAR
	diagnostic	value of the diag. byte of the CLEAR	CHAR
load	line_name[9]	name of X.25 line configuration	CHAR
	line.name.lenth	length of the line name parameter	WORD
openacc	connection_id	virtual circuit number	LONG
	facilities_len	length of the facility field	WORD
	facilities_data[128]	facility field	CHAR
	user_data_len	length of the user data	WORD
	user_data[128]	user data	CHAR
	D_bit_Services	delivery confirmation: 0 = no 1 = yes	WORD

Table 85. Verb Structure Variables (continued)

Function	Variable	Description	Type
openrec	match_first_byte	value of the 1st byte of user data	WORD
	first_byte_to_match	validate user data: 0 = no 1 = yes	BYTE
	d_bit_services	D_bit services requested: 0 = no 1 = yes	WORD
	calling_len	length of the calling PSDN address	WORD
	calling_address[8]	PSDN address of the calling DTE	CHAR
	called_len	length of the called PSDN address	WORD
	called_address[8]	PSDN address of the called DTE	CHAR
	timeout	delay to wait for an incoming CALL: -1 = indefinite	WORD
	facilities_len	length of the facility field	WORD
	facilities_data[128]	facility field	CHAR
	user_data_len	length of the user data	WORD
	user_data[128]	user data	CHAR
	connection_id	virtual circuit number	LONG
receive	connection_id	virtual circuit number	LONG
	buffer_len	size of buffer	WORD
	delay	data reception delay: -1 = infinite	WORD
	received_length	length of the received data	WORD
	buffer[BUFFER_SIZE]	data received	CHAR
	Q_bit	qualifier: 0 = normal data 1 = control data	WORD
	M_bit	multiple: 0 = no 1 = yes	WORD

Variable Definitions

The following variables are defined in this program:

Table 86. Variables Used in This Program

Variable	Description	Type
background	type of application indicator	BOOLEAN
*dest_file	destination file session number	FILE
ffirst	first packet indicator	BOOLEAN
bytes_written	no of bytes actually written	ULONG
totalexpected	no of bytes expected	ULONG
totalrcvd	no of bytes received	ULONG
rc	immediate return code	LONG

Program Structure

These steps show you how to run the code shown on page 326.

1. Check parameters. If no parameters, assume background application. First argument must be </f> or </F>.
2. Initialize the X.25 API.
3. Prepare to receive incoming.
4. Receive an incoming call.
5. Accept an incoming call.
6. Loop round receiving data until a Clear indication is received.
7. Begin loop.
8. Receive data on a virtual circuit. The first data packet contains the file information. The rest of the packets contain file data.
9. If first message:
 - Get and display files names and size of the receiving file.
 - Create destination file.
10. Rest of the messages: Write received data to destination file.
11. End loop.
12. Close the file.
13. Disconnect the virtual circuit.
14. End of the program.

Code

```
/*
    XNFCRCV.HIC
    Standard include files
*/
#include <stdio.h>           /* Standard include files */
#include <stdlib.h>
#include <string.h>
#undef NULL
#include <portabm.h>
/*
    X.25 Native Return Codes and function prototypes
*/
#include "adxhs51c.i86"      /* X.25 Native Return Codes */
#include "adxhs5kx.x86"      /* X.25 Native function prototypes */
/*
    Defines
*/
FILE *trc_file=NULL;
#define BUFFER_SIZE      4096      /* receive buffer size */
#define XPRINT1(A)        if (!background) { printf(A); } \
                           else \
                           { fprintf(trc_file,A); }
#define XPRINT2(A,B)      if (!background) { printf(A,B); } \
                           else \
                           { fprintf(trc_file,A,B); }
#define XPRINT3(A,B,C)    if (!background) { printf(A,B,C); } \
                           else \
                           { fprintf(trc_file,A,B,C); }
#define XPRINT4(A,B,C,D)  if (!background) { printf(A,B,C,D); } \
                           else \
                           { fprintf(trc_file,A,B,C,D); }
/*
    Structure for the first data packet
*/
```

```

typedef struct ft_data {
    CHAR src_file[60];
    CHAR dest_file[60];
    ULONG size;
} FT_DATA;

/*


Verb structures


*/

#define LINENAME        "X25LINE1"
#define LINENAME_LEN    8
struct load { CHAR line_name[9];
              WORD name_len; } Load = {LINENAME, LINENAME_LEN};

struct open_rec { WORD match_first_byte;
                  BYTE first_byte_to_match;
                  WORD d_bit_services;
                  WORD calling_len;
                  CHAR calling_address[8];
                  WORD called_len;
                  CHAR called_address[8];
                  WORD timeout;
                  WORD facilities_len;
                  CHAR facilities_data[128];
                  WORD user_data_len;
                  CHAR user_data[128];
                  LONG connection_id; } OpenRec = {0,NULL,0,"",0,"",-1};

struct open_acc { LONG connection_id;
                  WORD facilities_len;
                  CHAR facilities_data[128];
                  WORD user_data_len;
                  CHAR user_data[128];
                  WORD D_bit_Services; } OpenAcc = {0L,0,"",0,"",0};

struct receive { LONG connection_id;
                 WORD buffer_len;
                 WORD delay;
                 WORD received_length;
                 CHAR buffer[BUFFER_SIZE];
                 WORD Q_bit;
                 WORD M_bit; } Receive = {0L,BUFFER_SIZE,-1};

struct close { LONG connection_id;
               WORD facilities_len;
               CHAR facilities_data[128];
               WORD user_data_len;
               CHAR user_data[128];
               CHAR cause;
               CHAR diagnostic; } Close = {0,0,"",0,"",0,0};

/*


Function: main
Start of the receiver program.


*/

main(
    int    argc,
    char **argv)
{
/*


Variable declares and definitions


*/

LONG    rc;
ULONG   totalrcvd;
ULONG   totalexpected;

```

XNFCRCV C for X.25 API

```

ULONG    bytes_written;                /* No of bytes actually written */
BOOLEAN  ffirst = -1;
BOOLEAN  background = 0;
FILE     *dest_file;

/*
    Default is no match byte
*/

openrec.match_first_byte = 0;
switch (argc)
{
    case 0:
    case 1:
/*
    No parameters - assume background.
    Default match byte for background is '1'
*/

    background = -1;
    openrec.match_first_byte = 1;
    openrec.first_byte_to_match = (BYTE)1;
    if ((trc_file = fopen( "xnfcrcv.trc", "a+t" )) == NULL)
        return(1);
    XPRINT2("argc = %d\n", argc);
    break;
    case 2:
    case 3:
        if ( ( strcmp(argv[1],"/f") ) && ( strcmp(argv[1],"-f") ) &&
            ( strcmp(argv[1],"/F") ) && ( strcmp(argv[1],"-F") ) )
        {
            XPRINT1("First argument must be </f> or </F>.");
            return(1);
        }
        if (argc == 3)
        {
            if (strlen(argv[2]) > 1)
            {
                XPRINT1("Second argument must be 1 byte.");
                return(1);
            }
            openrec.match_first_byte = 1;
            openrec.first_byte_to_match = (char) atoi(argv[2]);
        }
        break;
    default:
        XPRINT1("Arguments incorrect: </f> and <match byte> only.");
        return(1);
}

/*
    We can now try to initialize the X.25 API.
    First call an internal function to set all the values in the
    control block and then call the X.25 function.
*/

XPRINT1("Initializing the X.25 API\n");
xload( Load.line_name,
        &Load.name_len,
        &rc);

if (rc != XA_OK)                /* Check completion return code */
{
    XPRINT2("Couldn't initialize X.25 API; return code = %lx\n",rc);
    return(1);
}
else
    XPRINT1("X.25 API Initialized\n");

```

```

/*
    Now we prepare to receive incoming calls
*/

xopenrec(&openrec.;match_first_byte,
        &openrec.;first_byte_to_match,
        &openrec.;WORD_d_bit_services,
        &openrec.;calling_len,
        openrec.calling_address,
        &openrec.;called_len,
        openrec.called_address,
        &openrec.;timeout,
        &openrec.;facilities_len,
        openrec.facilities_data,
        &openrec.;user_data_len,
        openrec.user_data,
        &openrec.;connection_id,
        &rc);

if (rc != XA_OK )      /* Check completion return code */
{
    XPRINT2("Listen failed, completion return code of %lx\n",rc);
    return(1);
}
else
    XPRINT1("About to issue a call accept\n");

openacc.connection_id = openrec.connection_id;
Receive.connection_id = openrec.connection_id;
Close.connection_id   = openrec.connection_id;

/*
    We now have an incoming call, so accept it.
*/

xopenacc(&openacc.connection_id,
        &openacc.facilities_len,
        openacc.facilities_data,
        &openacc.user_data_len,
        openacc.user_data,
        &openacc.D_bit_Services,
        &rc);

if (rc != XA_OK )      /* Check completion return code */
{
    XPRINT2("Call Accept failed, completion return code of %lx\n",rc);
    goto closeit;
}
else
    XPRINT1("Call Accepted OK\n");

/*
    Now loop round receiving data until a Clear Indication is received
*/

ffirst = TRUE;
totalrcvd = 0L;
totalexpected = 0L;
do
{
    xreceive(&Receive.connection_id;,
            &Receive.buffer_len;,
            &Receive.delay;,
            &Receive.received_length;,
            Receive.buffer,
            &Receive.Q_bit;,
            &Receive.M_bit;,
            &rc);

    if (rc != XA_OK )      /* Check completion return code */

```

XNFCRCV C for X.25 API

```
{
    XPRINT2("Data Receive failed, completion return code of %lx\n",rc);
    goto closeit;
}

/*


If this is the first data packet it contains the
        file information


*/
if (ffirst)
{
    XPRINT4("Receiving %ld bytes from %s to %s\n",
        ((FT_DATA *)Receive.buffer)->size,
        ((FT_DATA *)Receive.buffer)->src_file,
        ((FT_DATA *)Receive.buffer)->dest_file);

    totalexpected = ((FT_DATA *)Receive.buffer)->size;
    if ((dest_file =
        fopen(((FT_DATA *)Receive.buffer)->dest_file, "wb")) == NULL)
    {
        XPRINT2("Error Opening %s\n", ((FT_DATA *)Receive.buffer)->dest_file);
        return(1);
    }
    ffirst = FALSE;
    totalrcvd = 0L;
}
else
{
    bytes_written = fwrite( Receive.buffer,
                            1,
                            Receive.received_length,
                            dest_file );
    if (bytes_written != Receive.received_length)
    {
        XPRINT4("Error Writing to %s: expected=%d; written=%d\n",
            ((FT_DATA *)Receive.buffer)->dest_file,
            Receive.received_length, bytes_written);
        fclose(dest_file);
        goto closeit;
    }
    totalrcvd += Receive.received_length;
    XPRINT3("\rReceived %6.6ld bytes of %6.6ld", totalrcvd, totalexpected);
}

} while ((rc == XA_OK) && (totalrcvd < totalexpected));

/*


Close the file


*/
fclose(dest_file);
XPRINT1("\nFile Transfer Complete\n");
if (rc != XA_OK) /* if non data packet received */
{
    XPRINT1("Unexpected type of data has been received\n");
}
else
{
    XPRINT1("End of data Indicator received\n");
}

closeit:

    XPRINT1("Closing circuit...\n");

/*


Now that all data has been sent the call is cleared


*/
```

```

xclose(&Close.;connection_id,
      &Close.;facilities_len,
      Close.facilities_data,
      &Close.;user_data_len,
      Close.user_data,
      &Close.;cause,
      &Close.;diagnostic,
      &rc);

if (rc != XA_OK )      /* Check completion return code */
{
    XPRINT2("Clear failed, completion return code of %lx\n",rc);
    return(1);
}

XPRINT1("Call Cleared OK\n");

XPRINT1("Program finished\n");
if (background)
    fclose(trc_file);

return(0);              /* End program          */
}

```

XNFCSND.HIC File Transfer Send C Program for the X.25 API

This C program is used to send a file to a remote location. XNFCRCV may be used at the remote 4690.

The input parameters to this program are:

- Line name
- VCR name
- Source file
- Destination file
- Remote application name
- Call match byte

Remote application name and match byte are optional.

Include Files

The following include files are included in this program code:

stdio.h	Standard include files
stdlib.h	Standard include files
string.h	Standard include files
portabm.h	Standard include files
adxhs51c.i86	X.25 Native Return Codes
adxhs5kx.x86	X.25 Native function prototypes

Variables Used by Verb Structures

Table 87. Verb Structure Variables

Function	Variable	Description	Type
xclose	connection_id	virtual circuit number	LONG
	facilities_len	length of the facility field	WORD
	facilities_data[128]	facility field	CHAR
	user_data_len	length of the CLEAR user data	WORD
	user_data[128]	user data send with the CLEAR	CHAR
	cause	value of the cause byte of the CLEAR	CHAR
	diagnostic	value of the diag. byte of the CLEAR	CHAR
xevent	connection_id	SVC number	LONG
	event_type	type of event detected	WORD
	facilities_len	length of the facility field	WORD
	facilities_data[128]	facility field	CHAR
	data_len	length of data received	WORD
	data[128]	data received with the CLEAR or RESET	CHAR
	cause	value of the cause byte received with the CLEAR or RESET	CHAR
	diagnostic	value of the diagnostic byte received with the CLEAR or RESET	CHAR
xload	line_name[9]	name of X.25 line configuration	CHAR
	line.name.lenth	length of the line name parameter	WORD

Table 87. Verb Structure Variables (continued)

Function	Variable	Description	Type
xopen	vcr_name[9]	name of virtual circuit record	CHAR
	vcr_name_len	length of the user data	WORD
	appl_name[9]	name of remote 4680 or 4690 application	CHAR
	appl_name_len	length of the remote application	WORD
	D_bit_Services	delivery confirmation: 0 = no 1 = yes	WORD
	called_len	length of the called PSDN address	WORD
	called_address[8]	PSDN address of the called DTE	CHAR
	facilities_len	length of the facility field	WORD
	facilities_data[128]	facility field	CHAR
	user_data_len	length of user data	WORD
	user_data[128]	user data	CHAR
	connection_id	SVC number	LONG
xreceive	connection_id	virtual circuit number	LONG
	buffer_len	size of buffer	WORD
	delay	data reception delay: -1 = infinite	WORD
	received_length	length of the received data	WORD
	buffer[BUFFER_SIZE]	data received	CHAR
	Q_bit	qualifier: 0 = normal data 1 = control data	WORD
	M_bit	multiple: 0 = no 1 = yes	WORD
xsend	connection_id	SVC number	LONG
	data_len	length of the data already sent	WORD
	buffer[BUFFER_SIZE]	data to be sent	CHAR
	Q_bit	qualifier: 0 = normal 1 = control data	WORD
	M_bit	multiple: 0 = no 1 = yes	WORD
	D_bit	delivery confirmation: 0 = no 1 = yes	WORD
	data_len_sent	length of the data buffer	WORD

Variable Definitions

The following variables are defined in this program:

Table 88. Variables Used in This Program

Variable	Description	Type
BytesSent	number of bytes sent	ULONG
ffirst	indicator	BOOLEAN
err_text[128]	message text	FILE
destination file session number		FILE
sendfile[60]		CHAR
rcvfile[60]		CHAR
rc	immediate return code	LONG

Program Structure

These steps show you how to run the code shown on page 334.

1. Check number of parameters the program was called with, create default values for parameters, and use any parameters that are passed in.

If command line arguments, parse the parameters:

- Argument 1: VCR name
- Argument 2: Send file name
- Argument 3: Receive file name
- Argument 4: Remote application name
- Argument 5: Matching criteria "first byte"

2. Calculate the size of the file to send.
3. Open the file to send.
4. Initialize the X.25 API.
5. Make the call XOPEN.
6. Send the required number of data packets.

The packets will have the D-bit set, so that the verb will be completed only after the remote application has issued an acknowledgment.

7. Wait for CLEAR packet.
8. Ensure CLEAR has been received.
9. Close this end of the circuit.

Code

```

/*


XNFCSD.HIC
Standard include files


*/
#include <stdio.h> /* Standard include files */
#include <stdlib.h>
#include <string.h>
#undef NULL
#include <portabm.h>
/*


X.25 Native Return Codes and function prototypes


*/
#include "adxhs5lc.i86" /* X.25 Native Return Codes */
#include "adxhs5kx.x86" /* X.25 Native function prototypes */
/*


Defines


*/

```

```

#define BUFFER_SIZE      1024          /* buffer size for transmit data */
/*
    Structure for the first data packet
*/

typedef struct ft_data {
    CHAR src_file[60];
    CHAR dest_file[60];
    ULONG size;
} FT_DATA;
FT_DATA FileData;
/*CHAR buffer[BUFFER_SIZE];          /* temp buffer for sending data */
/*
    Prototypes for functions defined in this program
*/

extern void sys_err (CHAR *, CHAR *);
#define LINENAME          "X25LINE1"
#define LINENAME_LEN      8
#define RCV_TP            "XNFCRCV"
#define RCV_TP_LEN        7
/*
    Verb structures
*/

struct load { CHAR line_name[9];
              WORD name_len; } Load = {LINENAME, LINENAME_LEN};

struct open   { CHAR vcr_name[9];
              WORD vcr_name_len;
              CHAR appl_name[9];
              WORD appl_name_len;
              WORD D_bit_Services;
              WORD called_len;
              CHAR called_address[8];
              WORD facilities_len;
              CHAR facilities_data[128];
              WORD user_data_len;
              CHAR user_data[128];
              LONG connection_id; } Open = { "", 0, "", 0, 0, 0, "", 0, "", 0, "" };

struct receive { LONG connection_id;
                WORD buffer_len;
                WORD delay;
                WORD received_length;
                CHAR buffer[BUFFER_SIZE];
                WORD Q_bit;
                WORD M_bit; } Receive = { 0L, BUFFER_SIZE, -1, 0 };

struct send   { LONG connection_id;
              WORD data_len;
              CHAR buffer[BUFFER_SIZE];
              WORD Q_bit;
              WORD M_bit;
              WORD D_bit;
              WORD data_len_sent; } Send = { 0L, 0, "", 0, 0, 0, 0 };

struct close  { LONG connection_id;
              WORD facilities_len;
              CHAR facilities_data[128];
              WORD user_data_len;
              CHAR user_data[128];
              CHAR cause;
              CHAR diagnostic; } Close = { 0L, 0, "", 0, "", 0, 0 };

struct event  { LONG connection_id;
              WORD event_type;
              WORD facilities_len;

```

XNFCSD C for X.25 API

```
        CHAR facilities_data[128];
        WORD data_len;
        CHAR data[128];
        CHAR cause;
        CHAR diagnostic; } Event = { 0L };

/*
    Function: main
    Start of the transmitter program
*/

main(
    int  argc,
    char **argv)
{
/*
    Variable declares and definitions
*/

    CHAR    err_text[128];      /* Message text          */
    LONG    rc;                 /* Return code          */
    ULONG    BytesSent;         /* No of bytes sent     */
    BOOLEAN  fFirst = FALSE;
    FILE     *source_file;
    CHAR     sendfile[60];
    CHAR     rcvfile[60];

/*
    Check number of parameters the program was called with
*/

    if (argc < 2)
    {
        (void)printf(
            "Syntax is %s <vcr name> <source file> <dest file> <remote appl>,\n"
            "<Call match byte (1 - 9)>\n",argv[0]);
        return(1);
    }

/*
    Create Default values for parameters
*/

    strcpy(Open.vcr_name, "L1SVC1");
    Open.vcr_name_len = 6;

    strcpy(sendfile, "xsendfil");

    strcpy(rcvfile, "xrcvfil");

    strcpy(Open.appl_name, "");
    Open.appl_name_len = 0;

    Open.user_data_len = 0;
    Open.user_data[0] = 0;

/*
    Now use any parameters that are passed in.
*/

    if (argc > 1)
    {
        strcpy(Open.vcr_name, argv[1]);
        Open.vcr_name_len = strlen(Open.vcr_name);
    }

    if (argc > 2)
    {
        strcpy(sendfile, argv[2]);
    }
}
```

```

if (argc > 3)
{
    strcpy(rcvfile, argv[3]);
}

if (argc > 4)
{
    strcpy(Open.appl_name, argv[4]);
    Open.appl_name_len = strlen(Open.appl_name);
}

if (argc > 5)
{
    Open.user_data_len = 1;
    Open.user_data[0] = (char) atoi(argv[5]);
    (void)printf("Match byte = %x\n", Open.user_data[0]);
}

if ((source_file = fopen(sendfile, "rb")) == NULL)
{
    printf("Error Opening %s\n", sendfile);
    return(1);
}

/*


Calculate the size of the file


*/

if (fseek(source_file, 0L, SEEK_END))
{
    printf("Error seeking to end of file %s\n", sendfile);
    return(1);
}
FileData.size = ftell(source_file);
printf("Source file %s size = %ld\n", sendfile, FileData.size);
if (fseek(source_file, 0L, SEEK_SET))
{
    printf("Error seeking to start of file %s\n", sendfile);
    return(1);
}
strcpy(FileData.src_file, sendfile);
strcpy(FileData.dest_file, rcvfile);
BytesSent = 0L;

/*


We can now initialize the X.25 API.  

        First call an internal function to set all the values in the  

        control block and then call the X.25 function.


*/

(void)printf("Initializing the X.25 API (XLOAD)\n");
xload( Load.line_name,
        &Load.name_len;,
        &rc);
if (rc != XA_OK)
/* Check completion return code */
{
    (void)sprintf(err_text,
        "Couldn't initialize X.25 API, completion return code of %lx\n",
        rc);
    sys_err("XLOAD", err_text);
    return(1);
}
else
    (void)printf("X.25 API XLOAD done \n");

```

XNFCSD C for X.25 API

```
/*
    Now that the API is initialized, the call can be placed. Again
    an internal function is called to set up all the control information
    and then the API function is called. The program waits for the
    completion of the function. The call should now have been accepted
    by the remote application. The immediate and completion return code
    are checked. A special case is considered of the function failing
    because the call was cleared.
*/
*/

(void)printf("Making the call (XOPEN)\n");
xopen( Open.vcr_name,
        &Open.vcr_name_len;,
        Open.appl_name,
        &Open.appl_name_len;,
        &Open.D_bit_Services;,
        &Open.called_len;,
        Open.called_address,
        &Open.facilities_len;,
        Open.facilities_data,
        &Open.user_data_len;,
        Open.user_data,
        &Open.connection_id;,
        &rc);
if (rc != XA_OK )          /* Check completion return code */
{
    (void)sprintf(err_text,
                  "Couldn't make call, completion return code of %lx\n",
                  rc);
    sys_err("Open",err_text);
    return(1);
}
else
{
    Send.connection_id    = Open.connection_id;
    Receive.connection_id = Open.connection_id;
    Event.connection_id   = Open.connection_id;
    Close.connection_id   = Open.connection_id;
    (void)printf("Connection established to remote DTE\n");
}
/*
    Now send the required number of data packets. The packets will have
    the D-bit set, so that the wait on the semaphore will be completed
    only after the remote application has issued an acknowledgment.
*/
*/

printf("Sending %s to %s on VCR circuit %s\n", argv[2], argv[3], argv[1]);
fFirst = TRUE;
while (BytesSent < FileData.size)
{
    if (fFirst)
    {
        memcpy(Send.buffer, &FileData, sizeof(FileData));
        Send.data_len = sizeof(FileData);
        fFirst = FALSE;
    }
    else
    {
        if ((Send.data_len = fread(Send.buffer,
                                   1,
                                   sizeof(Send.buffer),
                                   source_file)) < 0)
        {
            printf("Error reading %s rc = %ld\n",
                  FileData.src_file, Send.data_len);
            return(1);
        }
        BytesSent += Send.data_len;
    }
}
```

```

    }
    xsend(&Send.connection_id,
          &Send.data_len,
          Send.buffer,
          &Send.Q_bit,
          &Send.M_bit,
          &Send.D_bit,
          &Send.data_len_sent,
          &rc);

    if (rc != XA_OK )      /* Check completion return code */
    {
        (void)sprintf(err_text,
                      "Send failed, completion return code of %lx\n",
                      rc);
        sys_err("Send",err_text);
        return(1);
    }
    else
        (void)printf("Sent %6.6ld bytes of %6.6ld\r",BytesSent, FileData.size);
}
(void)printf("\nSent data OK. Waiting for CLEAR packet...\n");
fclose(source_file);

/*


Now wait for clear packet


*/

do
{
    xreceive(&Receive.connection_id,
             &Receive.buffer_len;,
             &Receive.delay;,
             &Receive.received_length;,
             Receive.buffer,
             &Receive.Q_bit;,
             &Receive.M_bit;,
             &rc);

    if (rc != XA_EVENT )      /* Check completion return code */
    {
        (void)sprintf(err_text,
                      "Unexpected RC from Receive rc = %lx\n",
                      rc);
        sys_err("Receive",err_text);
        return(1);
    }

    (void)printf("XEVENT...\n");

/*


Ensure clear has been received


*/

    xevent(&Event.connection_id,
           &Event.event_type,
           &Event.facilities_len,
           Event.facilities_data,
           &Event.data_len,
           Event.data,
           &Event.cause,
           &Event.diagnostic,
           &rc);
    if (rc != XA_OK)          /* Check immediate return code */
    {
        (void)sprintf(err_text,
                      "Error receiving Event, return code of %lx\n",rc);
        sys_err("Event",err_text);
        return(1);
    }
}

```

XNFCSD C for X.25 API

```
else if (Event.event_type == EVENT_CLEAR)
{
    (void)printf("Clear indication packet received\n");

    (void)printf("XCLOSE...\n");

/*


Close this end of the circuit


*/

    xclose(&Close.;connection_id,
          &Close.;facilities_len,
          Close.facilities_data,
          &Close.;user_data_len,
          Close.user_data,
          &Close.;cause,
          &Close.;diagnostic,
          &rc);

    if (rc != XA_OK )          /* Check completion return code */
    {
        (void)sprintf(err_text,
                      "Clear failed, completion return code of %lx\n",
                      rc);
        sys_err("Close",err_text);
        return(1);
    }
}
else
{
    (void)printf("Unexpected type of data has been received\n");
}

} while (Event.event_type != EVENT_CLEAR); /* until a clear pkt received */

(void)printf("Program finished successfully\n");

return(0);                      /* End program */

/*


Function to print an error message to stderr, tidy up and exit


*/

static void sys_err(
    CHAR *verb,          /* Verb identifier string */
    CHAR *errmsg)        /* Error message */
{
    (void)fprintf(stderr,"%s: %s", verb, errmsg);
    /* Print the error message */
    return;
    /* End the program */
}
```

Include Files for C X.25 Application Programs

This section contains the include files to be used with C X.25 application programs that call the operating system X.25 API.

X.25 Include File for C: ADXHS5KX.X86

This file, which is on the 4690 Optionals, contains external statements defining the reference and parameters of all subroutines that are referenced by C programs using the X.25 API.

X.25 API verb prototype file for C

XLOAD

```
extern void xload ( char *line_name,
                   WORD *line_name_length,
                   LONG *return_code );
```

XSTO

```
extern void xsto ( LONG *verb_timeout_seconds,
                  LONG *return_code );
```

XOPEN

```
extern void xopen ( char *vcr_name,
                   WORD *vcr_name_length,
                   char *remote_application,
                   WORD *remote_application_length,
                   WORD *D_bit_services,
                   WORD *called_address_length,
                   char *called_address,
                   WORD *facilities_field_length,
                   char *facilities_field,
                   WORD *call_user_data_length,
                   char *call_user_data,
                   LONG *connection_id,
                   LONG *return_code );
```

XSEND

```
extern void xsend ( LONG *connection_id,
                   WORD *data_length,
                   char *data,
                   WORD *Q_bit_indicator,
                   WORD *M_bit_indicator,
                   WORD *D_bit_indicator,
                   WORD *data_bytes_sent,
                   LONG *return_code );
```

XRECEIVE

```
extern void xreceive ( LONG *connection_id,
                      WORD *data_buffer_size,
                      WORD *call_receipt_delay,
                      WORD *data_length,
                      char *data,
                      WORD *Q_bit_indicator,
                      WORD *M_bit_indicator,
                      LONG *return_code );
```

X.25 C Include File

XCLOSE

```
extern void xclose ( LONG *connection_id,
                    WORD *facilities_field_length,
                    char *facilities_field,
                    WORD *clear_user_data_length,
                    char *clear_user_data,
                    char *cause_code,
                    char *diagnostic_code,
                    LONG *return_code );
```

XEVENT

```
extern void xevent ( LONG *connection_id,
                    WORD *event_type,
                    WORD *facilities_field_length,
                    char *facilities_field,
                    WORD *data_length,
                    char *data,
                    char *cause_code,
                    char *diagnostic_code,
                    LONG *return_code );
```

XOPENREC

```
extern void xopenrec ( WORD *first_byte_of_user_data,
                      char *first_byte_of_user_data_value,
                      WORD *D_bit_services,
                      WORD *calling_address_length,
                      char *calling_address,
                      WORD *called_address_length,
                      char *called_address,
                      WORD *call_receipt_delay,
                      WORD *facilities_field_length,
                      char *facilities_field,
                      WORD *call_user_data_length,
                      char *call_user_data,
                      LONG *connection_id,
                      LONG *return_code );
```

XOPENACC

```
extern void xopenacc ( LONG *connection_id,
                      WORD *facilities_field_length,
                      char *facilities_field,
                      WORD *call_user_data_length,
                      char *call_user_data,
                      WORD *D_bit_services,
                      LONG *return_code );
```

XIT

```
extern void xit      ( LONG *connection_id,
                      WORD *interrupt_data_length,
                      char *interrupt_data,
                      LONG *return_code );
```

XRESET

```
extern void xreset   ( LONG *connection_id,
                      char *cause_code,
                      char *diagnostic_code,
                      LONG *return_code );
```

X.25 C Declaration Include File

XALLOC

```
extern void xalloc ( char *vcr_name,  
                    WORD *vcr_name_length,  
                    LONG *connection_id,  
                    LONG *return_code );
```

XDEALLOC

```
extern void xdealloc ( LONG *connection_id,  
                      LONG *return_code );
```

XWAIT

```
extern void ADX_CWAIT ( LONG *ret,  
                       LONG wtime,  
                       UWORD wcount,  
                       void *wblk );
```

S_OPEN

```
extern LONG S_OPEN ( WORD flags, BYTE *name );
```

S_CLOSE

```
extern LONG S_CLOSE ( WORD flags, LONG fnum );
```

S_SPECIAL

```
extern LONG S_SPECIAL ( BYTE func, WORD flags, LONG fnum,  
                      BYTE *dbuf, LONG dbufsize,  
                      BYTE *pbuf, LONG pbufsize );
```

X.25 Verb Return Code Declaration File: ADXHS5LC.I86

This is the X.25 verb return code declaration file, which is on the 4690 Optionals. It should be included by C programs. Table 89 on page 347 specifies the return codes.

Event types returned by the XEVENT verb

```
#define EVENT_CLEAR          1
#define EVENT_RESET          2
#define EVENT_INTERRUPT      3
```

X.25 constants

```
#define MAX_LINENAME_LEN     8      /* physical line name */
#define MAX_VCRNAME_LEN     8      /* circuit config record */
#define MAX_ADDRESS_LEN     15     /* calling & called addr */
#define MAX_APPL_LEN        8      /* remote appl name */
#define MAX_DATA             32767L /* send & receive data */
#define MAX_INT80            1      /* interrupt data 1980 */
#define MAX_INT84            32     /* interrupt data 84/88 */
#define MAX_USER_DATA        16     /* call & clear user data */
#define MAX_USER_DATA_FAST   128    /* fast select */
#define MAX_USER_DATA_EVENT  128    /* event user data */
#define MAX_FAC80            63     /* facilities data 1980 */
#define MAX_FAC84            109    /* facilities data 84/88 */
#define MAX_PACKET_LEN       128    /* packet size */
#define MAX_CAUSE_CODE_LEN   1      /* cause code */
#define MAX_DIAG_CODE_LEN    1      /* diagnostic code */
```

Verb constants

```
#define XNOVERB_c            0
#define XLOAD_c              1
#define XSTO_c               2
#define XOPEN_c              3
#define XSEND_c              4
#define XRECEIVE_c           5
#define XCLOSE_c             6
#define XEVENT_c             7
#define XOPENREC_c           8
#define XOPENACC_c           9
#define XIT_c                10
#define XRESET_c             11
#define XALLOC_c             12
#define XDEALLOC_c           13
#define XWAIT_c              14
#define XWAITCAN_c           15
#define XOPENXUI_c           101
#define XQUERY_c             102
```

Compiling and Linking the X.25 API Programs

The BASIC file transfer sample programs are named XNFBSND.BAS (file requester) and XNFBRCV.BAS (file server). Compile them on the 4690 store controller using the 4680 BASIC compiler. See the *4680 Basic: Language Reference* for compiler options and details.

When compiling, be sure that the three include files are in the same subdirectory as the program. The include files are contained on the X.25 API diskette and their contents are as follows:

ADXHS5KF.J86	Procedural interface
ADXHS5LF.J86	Declaration of return codes and variables
ADXHS5MF.J86	Initialization of return codes

Compiling and Linking X.25 Programs

These include files are provided to make the program more readable and consistent with the parameter names specified in Chapter 13, “X.25 API Verb Reference,” on page 235. It is recommended that these files be used when developing X.25 application programs written in 4680 BASIC.

The compile command for the file requester program is:

```
BASIC XNFBSDND [BU]
```

Linking

The verb library for 4680 BASIC, ADXHS51L.L86, must be linked with the file transfer program object code. The format of the link file XNFSND.INP for the file requester program is:

```
XNFBSDND=XNFBSDND,  
                ADXHS51L.L86 [S],  
                ADXACRCL.L86 [S,LI,LO,M,NOSH,STACK[MAX[65535]]]
```

The link command for the file requester program is:

```
LINK86 XNFBSDND [I]
```

Using this command for both the file requester and the file server generates XNFBSDND.286 and XNFBRCV.286. For this example, these programs are located in the ADX_UPGM subdirectory or in the subdirectory used for compiling and editing programs.

Starting the Requester for X.25 Programs

To start the requester program in foreground mode, provide the following information in command mode:

- XNFBSDND (program name)
- File to be transferred
- File name (target)
- Remote program name
- Other parameters

Starting the Server for X.25 Programs

There are three ways to start the remote program:

- In command mode as a foreground task, type the following:

```
XNFBRCV \f
```

and press **Enter**. The program starts and waits.

- By a request from the requester program as a background task. When you start the requester program (XNFBSDND) provide the name of the remote program (XNFBRCV) as the third parameter.
- By the communication driver (X.25 API) as a background task. When configuring the VCR, specify the remote application name.

For a background application, the communications driver (X.25 API) must be loaded before starting the application. For a foreground application, if the communications driver is not loaded, the program loads it.

The 8-character remote application name is a logical name that must be defined on the 4690 with its full path and executable name.

Appendix D. X.25 API Reference Information

This appendix provides reference information for the X.25 verbs supported by the X.25 API. For information about designing operating system applications that call the X.25 API verbs, see Chapter 12, “Designing an X.25 Application,” on page 229. For a description of each supported X.25 API verb and its parameters, see Chapter 13, “X.25 API Verb Reference,” on page 235. For examples of application programs that call X.25 API verbs, see Appendix C, “Examples of X.25 API Programs and Transaction Programs,” on page 293. (Configuration information for the examples is in the *4690 OS: Planning, Installation, and Configuration Guide*.)

X.25 Verb Return Codes

Table 89 provides a list of return codes returned by the X.25 API.

Table 89. X.25 Verb Return Codes and Descriptions

BASIC Initialization	C Declaration	Value	Meaning
XA.OK	XA_OK	0	Verb successful.
XA.ERR.ADAPTER	XA_ERR_ADAPTER	80B21280	Line error on adapter. Negative or no response to XLOAD that was sent to initialize adapter.
XA.ERR.RESOURCE	XA_ERR_RESOURCE	80B21281	Failure to allocate system resource such as memory, flags, event masks.
XA.ERR.LINE.NOT.LOADED	XA_ERR_LINE_NOT_LOADED	80B21282	An attempt was made to XOPEN SVC or XALLOC PVC on a line that has not been XLOADED.
XA.PVC.INUSE	XA_PVC_INUSE	80B21290	An attempt was made to XALLOC a PVC that is already in use.
XA.NO.SVC.AVAILABLE	XA_NO_SVC_AVAILABLE	80B21291	No more SVCs are available.
XA.NOTHING.RECEIVED.IN.DELAY	XA_NOTHING_RECEIVED_IN_DELAY	80B21292	The receipt delay for XRECEIVE or XOPENREC has been exceeded.
XA.TIMEOUT	XA_TIMEOUT	80B21293	The XSTO timeout value has been exceeded on a verb other than XRECEIVE or XOPENREC. A zero timeout value was in force for one of the verbs XSEND, XIT, XRESET or XOPENACC.

Table 89. X.25 Verb Return Codes and Descriptions (continued)

BASIC Initialization	C Declaration	Value	Meaning
XA.IN.PROGRESS	XA_IN_PROGRESS	80B21294	Attempt to XLOAD a line while another XLOAD is in progress for that line.
XA.EVENT	XA_EVENT	80B21295	An event (Clear, Reset, or Interrupt) was received before the current verb completed.
XA.NO.EVENT	XA_NO_EVENT	80B21296	Returned by XEVENT if no event is outstanding.
XA.INVALID.RESPONSE.XSEND	XA_INVALID_RESPONSE_XSEND	80B21297	Returned by XSEND if, in response to a data packet, anything other than a normal acknowledgment or a Clear, Reset, or Interrupt was received.
XA.INVALID.RESPONSE.XCLOSE	XA_INVALID_RESPONSE_XCLOSE	80B21298	Returned by XCLOSE if, in response to a Clear request, anything other than a Clear Confirm, Clear, Reset, or Interrupt was received.
XA.INVALID.RESPONSE.XRESET	XA_INVALID_RESPONSE_XRESET	80B21299	Returned by XRESET if, in response to a Reset request, anything other than a Reset Confirm, Clear, Reset, or Interrupt was received.
XA.INVALID.RESPONSE.XIT	XA_INVALID_RESPONSE_XIT	80B2129A	Returned by XIT if, in response to an Interrupt, anything other than an Interrupt Confirm, Clear, Reset, or Interrupt was received.
XA.TIMEOUT.CALL.CONNECT	XA_TIMEOUT_CALL_CONNECT	80B2129B	Returned by XOPEN if an expected Call Connect was not received.

Table 89. X.25 Verb Return Codes and Descriptions (continued)

BASIC Initialization	C Declaration	Value	Meaning
XA.ERR.PARAMETER	XA_ERR_PARAMETER	80B212A0	The user application has specified a parameter to a verb that is not addressable by the API, probably due to specifying a buffer that is shorter than the associated minimum length or shorter than the specified length.
XA.ERR.CONNECTION.ID	XA_ERR_CONNECTION_ID	80B212A1	Invalid connection ID.
XA.ERR.TIMEOUT	XA_ERR_TIMEOUT	80B212A2	Not returned.
XA.ERR.D.BIT	XA_ERR_D_BIT	80B212A3	Returned by XSEND on SVCs only if the delivery confirmation indicator (D-bit) was requested but D-bit services had not been successfully established by XOPEN or XOPENACC.
XA.ERR.M.BIT	XA_ERR_M_BIT	80B212A4	Returned by XSEND if the user requested the more data mark (M-bit) but the sending data size was not a multiple of 128 bytes.
XA.RESERVED1	XA_RESERVED1	80B212A5	Reserved.
XA.ERR.CALLED.LENGTH	XA_ERR_CALLED_LENGTH	80B212A6	Returned by XOPENREC or XOPEN if the specified called address length is not valid (negative or greater than 15).
XA.ERR.CALLING.LENGTH	XA_ERR_CALLING_LENGTH	80B212A7	Returned by XOPENREC if the specified calling address length is not valid (negative or greater than 15).

Table 89. X.25 Verb Return Codes and Descriptions (continued)

BASIC Initialization	C Declaration	Value	Meaning
XA.ERR.DATA.LENGTH	XA_ERR_DATA_LENGTH	80B212A8	Returned by XOPEN, XCLOSE, XSEND, XRECEIVE or XIT if the specified data length (call user data, Clear user data, Send data, or Interrupt data) is not valid (negative or greater than the maximum allowable).
XA.ERR.FACILITIES.LENGTH	XA_ERR_FACILITIES_LENGTH	80B212A9	Returned by XOPEN or XCLOSE if the specified facilities field length is not valid (negative or greater than 109).
XA.ERR.VCR.NAME.LENGTH	XA_ERR_VCR_NAME_LENGTH	80B212AA	Returned by XOPEN or XALLOC if the specified VCR name length is not valid (negative or greater than 8).
XA.RESERVED2	XA_RESERVED2	80B212AB	Reserved.
XA.ERR.REMOTE.APPL.LENGTH	XA_ERR_REMOTE_APPL_LENGTH	80B212AC	Returned by XOPEN if the specified remote application name length is not valid (negative or greater than 8).
XA.ERR.RECEIPT.DELAY	XA_ERR_RECEIPT_DELAY	80B212AD	Returned by XRECEIVE or XOPENREC if specified receipt delay is not valid (less than -1).
XA.ERR.CONFIGURATION	XA_ERR_CONFIGURATION	80B212B0	Returned by XOPEN or XALLOC if the API is unable to read either the circuit configuration record or the associated line configuration record.
XA.ERR.NOT.PVC	XA_ERR_NOT_PVC	80B212B1	Attempt to XALLOC a circuit that is not a PVC.

Table 89. X.25 Verb Return Codes and Descriptions (continued)

BASIC Initialization	C Declaration	Value	Meaning
XA.ERR.NOT.SVC	XA_ERR_NOT_SVC	80B212B2	Attempt to XOPEN a circuit that is not an SVC.

Note: The BASIC initialization of the return codes should be a 4-byte integer

X.25 Communications Return Codes

Table 90 provides a list of system return codes returned only by X.25 API.

Table 90. X.25 Communications Return Codes

Value	Meaning
80B305F5	<p>ERR Code = xx</p> <p>Explanation: X.25 is active.</p> <p>The last application using an SNA X.25 link has closed and the link was configured as non-resident (OR Disable Link for an SNA X.25 link has been issued), but there are still X.25 API circuits active, so the communications driver has not been uninstalled.</p> <p>User Response:</p> <p>No action needed; the communications driver will be uninstalled when the X.25 circuit becomes inactive.</p>
80B305F6	<p>ERR Code = xx</p> <p>Explanation: This is an internal X.25 API error.</p> <p>User Response:</p> <p>Initiate a store controller dump and contact the Toshiba Support Center for assistance.</p>

X.25 Packet Type Identifier

from DCE to DTE	from DTE to DCE	8	7	6	5	4	3	2	1	dec	hex
call set up and clearing											
Incoming Call	Call request	0	0	0	0	1	0	1	1	011	0B
Call Connected	Call accepted	0	0	0	0	1	1	1	1	015	0F
Clear Indication	Clear request	0	0	0	1	0	0	1	1	019	13
DCE Clear Conf.	DTE Clear conf.	0	0	0	1	0	1	1	1	023	17
data and interrupt											
DCE data	DTE data	x	x	x	x	x	x	x	0
DCE interrupt	DTE interrupt	0	0	1	0	0	0	1	1	035	23
DCE interrupt conf.	DTE interrupt conf.	0	0	1	0	0	1	1	1	039	27
flow control and reset											
DCE RR (modulo 8)	DTE RR(modulo 8)	x	x	x	0	0	0	0	1	..1	.1
DCE RNR (modulo 8)	DTE RNR(modulo 8)	x	x	x	0	0	1	0	1	..5	.5
	DTE REJ(modulo 8)	x	x	x	0	1	0	0	1	..9	.9
Reset indication	Reset request	0	0	0	1	1	0	1	1	027	1B
DCE reset conf.	DTE rest conf.	0	0	0	1	1	1	1	1	031	1F
restart											
Restart indication	Restart request	1	1	1	1	1	0	1	1	251	FB
DCE restart conf.	DTE restart conf.	1	1	1	1	1	1	1	1	255	FF
diagnostic											
Diagnostic		1	1	1	1	0	0	0	1	241	F1
registration											
	Registration request	1	1	1	1	0	0	1	1	243	F3
Registration conf.		1	1	1	1	0	1	1	1	247	F7

Note: A bit that is indicated x can be set to either 0 or 1.

Note: The operating system does not support modulo 128.

Figure 31. X.25 Packet Type Identifier

X.25 Constants

MAX_LINENAME_LEN	8	physical line name
MAX_VCRNAME_LEN	8	circuit config record
MAX_ADDRESS_LEN	15	calling & called addr
MAX_APPL_LEN	8	remote appl name
MAX_DATA	32767L	send & receive data
MAX_INT80	1	interrupt data 1980
MAX_INT84	32	interrupt data 84/88
MAX_USER_DATA	16	call & clear user data
MAX_USER_DATA_FAST	128	fast select
MAX_USER_DATA_EVENT	128	event user data
MAX_FAC80	63	facilities data 1980
MAX_FAC84	109	facilities data 84/88
MAX_PACKET_LEN	128	packet size
MAX_CAUSE_CODE_LEN	1	cause code
MAX_DIAG_CODE_LEN	1	diagnostic code

Figure 32. X.25 Constants

X.25 Event Types

The following event types are returned by the XEVENT verb

Mnemonic	Value
EVENT_CLEAR	1
EVENT_RESET	2
EVENT_INTERRUPT	3

Figure 33. Event Types

X.25 Cause Codes

REGISTRATION cause code	8	7	6	5	4	3	2	1	Dec	Hex
Invalid facility request	0	0	0	0	0	0	1	1	3	x03
Network congestion	0	0	0	0	0	1	0	1	5	x05
Local procedure error	0	0	0	1	0	0	1	1	19	x13
Registration/cancel confirmed	0	1	1	1	1	1	1	1	127	x7F
RESTART cause code	8	7	6	5	4	3	2	1	Dec	Hex
Remote procedure error	0	0	0	0	0	0	0	1	1	x01
Network congestion	0	0	0	0	0	0	1	1	3	x03
Network operational	0	0	0	0	0	1	1	1	7	x07
Registration/cancel confirmed	0	1	1	1	1	1	1	1	127	x7F
RESET cause code	8	7	6	5	4	3	2	1	Dec	Hex
DTE originated	0	0	0	0	0	0	0	0	0	x00
DTE originated	1	x	x	x	x	x	x	x	128	x80
Out of order	0	0	0	0	0	0	0	1	1	x01
Remote procedure error	0	0	0	0	0	0	1	1	3	x03
Local procedure error	0	0	0	0	0	1	0	1	5	x05
Network congestion	0	0	0	0	0	1	1	1	7	x07
Remote DTE operational	0	0	0	0	1	0	0	1	9	x09
Network operational	0	0	0	0	1	1	1	1	15	x0F
Incompatible destination	0	0	0	1	0	0	0	1	17	x11
Network out of order	0	0	0	1	1	1	0	1	29	x1D
CLEAR cause code	8	7	6	5	4	3	2	1	Dec	Hex
DTE originated	0	0	0	0	0	0	0	0	0	x00
DTE originated	1	x	x	x	x	x	x	x	128	x80
Number busy	0	0	0	0	0	0	0	1	1	x01
Invalid facility request	0	0	0	0	0	0	1	1	3	x03
Network congestion	0	0	0	0	0	1	0	1	5	x05
Out of order	0	0	0	0	1	0	0	1	9	x09
Access barred	0	0	0	0	1	0	1	1	11	x0B
Not obtainable	0	0	0	0	1	1	0	1	13	x0D
Remote procedure error	0	0	0	1	0	0	0	1	17	x11
Local procedure error	0	0	0	1	0	0	1	1	19	x13
RPOA out of order	0	0	0	1	0	1	0	1	21	x15
Reverse charge not subscribed	0	0	0	1	1	0	0	1	25	x19
Incompatible destination	0	0	1	0	0	0	0	1	33	x21
Fast select not subscribed	0	0	1	0	1	0	0	1	41	x29
Ship absent (mobile maritime)	0	0	1	1	1	0	0	1	57	x39

Note: DTE-originated cause codes can be 0, 128 or greater.

Figure 34. X.25 Cause Codes

X.25 Network-Generated Diagnostic Codes

DIAGNOSTIC (Network generated)	8	7	6	5	4	3	2	1	Dec	Hex
No additional information	0	0	0	0	0	0	0	0	0	x00
Invalid P(S)	0	0	0	0	0	0	0	1	1	x01
Invalid P(R)	0	0	0	0	0	0	1	0	2	x02
Packet type invalid	0	0	0	1	0	0	0	0	16	x10
For state r1	0	0	0	1	0	0	0	1	17	x11
etc. for r2-r3,p1-p7,d1-d2	0	0	0	1	0	x	x	x	-	x1-
For state d3	0	0	0	1	1	1	0	1	29	x1D
Packet type not allowed	0	0	1	0	0	0	0	0	32	x20
Unidentifiable packet	0	0	1	0	0	0	0	1	33	x21
Call on one-way channel	0	0	1	0	0	0	1	0	34	x22
Invalid packet type on a PVC	0	0	1	0	0	0	1	1	35	x23
Packet on unassigned channel	0	0	1	0	0	1	0	0	36	x24
Reject not subscribed to	0	0	1	0	0	1	0	1	37	x25
Packet too short	0	0	1	0	0	1	1	0	38	x26
Packet too long	0	0	1	0	0	1	1	1	39	x27
Invalid GFI	0	0	1	0	1	0	0	0	40	x28
Restart/Registration error	0	0	1	0	1	0	0	1	41	x29
Packet type not comp. with facility	0	0	1	0	1	0	1	0	42	x2A
Unauthorized interrupt confirmation	0	0	1	0	1	0	1	1	43	x2B
Unauthorized interrupt	0	0	1	0	1	1	0	0	44	x2C
Unauthorized reject	0	0	1	0	1	1	0	1	45	x2D
Time expired	0	0	1	1	0	0	0	0	48	x30
For incoming call	0	0	1	1	0	0	0	1	49	x31
For CLEAR indication	0	0	1	1	0	0	1	0	50	x32
For RESET indication	0	0	1	1	0	0	1	1	51	x33
For restart indication	0	0	1	1	0	1	0	0	52	x34
For call deflection	0	0	1	1	0	1	0	1	53	x35
Call setup/clear/reg problem	0	1	0	0	0	0	0	0	64	x40
Facility/registration illegal	0	1	0	0	0	0	0	1	65	x41
Facility parameter not allowed	0	1	0	0	0	0	1	0	66	x42
Invalid called DTE address	0	1	0	0	0	0	1	1	67	x43
Invalid calling DTE address	0	1	0	0	0	1	0	0	68	x44
Invalid facility/registration len	0	1	0	0	0	1	0	1	69	x45
Incoming call barred	0	1	0	0	0	1	1	0	70	x46
No logical channel available	0	1	0	0	0	1	1	1	71	x47
Call collision	0	1	0	0	1	0	0	0	72	x48
Duplicate facility requested	0	1	0	0	1	0	0	1	73	x49
Non-zero address length	0	1	0	0	1	0	1	0	74	x4A
Non-zero facility length	0	1	0	0	1	0	1	1	75	x4B
Facility not provided but expected	0	1	0	0	1	1	0	0	76	x4C
Invalid CCITT DTE facility	0	1	0	0	1	1	0	1	77	x4D
Max call redirections/deflections	0	1	0	0	1	1	1	0	78	x4E

Figure 35. X.25 Network-Generated Diagnostic Codes: Part A

Miscellaneous	0	1	0	1	0	0	0	0	80	x50
Improper cause code from DTE	0	1	0	1	0	0	0	1	81	x51
Not aligned octet	0	1	0	1	0	0	1	0	82	x52
Inconsistent Q bit setting	0	1	0	1	0	0	1	1	83	x53
NUI problem	0	1	0	1	0	1	0	0	84	x54
International problem	0	1	1	1	0	0	0	0	112	x70
Network specific diagnostic start	1	0	0	0	0	0	0	0	128	x80
Network specific diagnostic end	1	1	1	1	1	1	1	1	255	xFF

Figure 36. X.25 Network-Generated Diagnostic Codes: Part B

DTE-Generated Diagnostic Codes

DIAGNOSTIC (if Cause code x'80' or x'00')	8	7	6	5	4	3	2	1	Dec	Hex
No additional information	0	0	0	0	0	0	0	0	0	x00
Invalid packet	0	0	0	1	0	0	0	0	16	x10
Invalid R1	0	0	0	1	0	0	0	1	17	x11
Invalid R3	0	0	0	1	0	0	1	0	18	x12
Invalid R2	0	0	0	1	0	0	1	1	19	x13
Invalid P1	0	0	0	1	0	1	0	0	20	x14
Invalid P3	0	0	0	1	0	1	0	1	21	x15
Invalid P2	0	0	0	1	0	1	1	0	22	x16
Invalid P4	0	0	0	1	0	1	1	1	23	x17
Invalid P5	0	0	0	1	1	0	0	0	24	x18
Invalid P7	0	0	0	1	1	0	0	1	25	x19
Invalid P6	0	0	0	1	1	0	1	0	26	x1A
Invalid D1	0	0	0	1	1	0	1	1	27	x1B
Invalid D3	0	0	0	1	1	1	0	0	28	x1C
Invalid D2	0	0	0	1	1	1	0	1	29	x1D
Timer exceeded	0	0	1	1	0	0	0	0	48	x30
Timeout on call	0	0	1	1	0	0	0	1	49	x31
Timeout on CLEAR	0	0	1	1	0	0	1	0	50	x32
Timeout on RESET	0	0	1	1	0	0	1	1	51	x33
Timeout on restart	0	0	1	1	0	1	0	0	52	x34
Unspecified call setup problem	0	1	0	0	0	0	0	0	64	x40
Invalid called length	0	1	0	0	0	0	0	0	64	x40
Invalid calling length	0	1	0	0	0	0	0	0	64	x40
Invalid user data length	0	1	0	0	0	0	0	0	64	x40
Invalid called address	0	1	0	0	0	0	1	1	67	x43
Invalid calling address	0	1	0	0	0	1	0	0	68	x44
Link down	1	0	0	0	0	0	1	1	131	x83
Invalid PVC	1	0	1	0	0	0	1	1	163	xA3
Packet too short	1	0	1	0	0	1	1	0	166	xA6
Packet too long	1	0	1	0	0	1	1	1	167	xA7
Facilities too long	1	0	1	0	0	1	1	1	167	xA7
Invalid GFI	1	0	1	0	1	0	0	0	168	xA8
Unidentified packet	1	0	1	0	1	0	0	1	169	xA9
Not compatible	1	0	1	0	1	0	1	0	170	xAA
Invalid P(S)	1	0	1	0	1	0	1	1	171	xAB
Invalid P(R)	1	0	1	0	1	1	0	0	172	xAC
No D bit	1	0	1	0	1	1	0	1	173	xAD
Inconsistent Q bit	1	0	1	0	1	1	1	0	174	xAE
Uninitialized channel	1	1	0	0	0	0	1	1	195	xC3
Uninitialized	1	1	0	0	0	1	0	0	196	xC4
Invalid restart	1	1	1	0	0	0	1	0	226	xE2
Incoming call on outgoing channel	1	1	1	0	0	0	1	1	227	xE3
Invalid facilities parameters	1	1	1	0	0	1	1	0	230	xE6
Invalid facilities code	1	1	1	0	0	1	1	1	231	xE7
Application not waiting for call	1	1	1	1	0	0	0	1	241	xF1
Application cancelled	1	1	1	1	0	0	1	0	242	xF2

Figure 37. DTE-Generated Diagnostic Codes

DTE Addressing

See Figure 38 on page 357 for the DTE addressing network user address format.

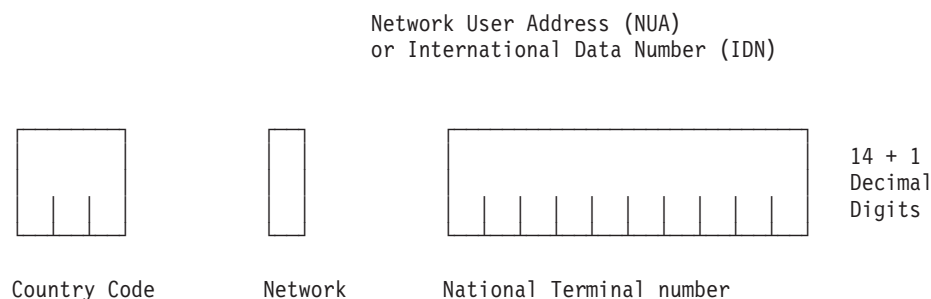


Figure 38. DTE Addressing Network User Address

The first four digits of the International Data Network (IDN) are called the data network identification code (DNIC).

The first three digits indicate in which the country the PSDN is located. The fourth digit identifies the PSDN in that country.

The DTE within a network is identified by a National Terminal Number (NTN), which can be up to ten digits long.

Country code	Country	DNIC	Network
204	Netherlands	2044	DABAS
208	France	2080	Transpac
214	Spain	214.	Iberpac
222	Italy	222.	
234	United Kingdom	2341	IPSS
234		2342	PSS
234		234.	Mercury
238	Denmark	2383	Euronet
240	Sweden	2405	Telepak
242	Norway	2422	Norpak
262	Germany	2624	Datex-P
272	Ireland	2721	PTT
274	Iceland	2740	Icepak
302	Canada	3020	Datapac
302		3025	Teleglobe
302		3029	Infoswitch
310	United States	3106	Tymnet
311		3110	Telenet
312		3125	Uninet
312		3126	Autonet
313		3137	Infonet
440	Japan	4401	DDX-P
440		4408	Venus-P

Figure 39. DTE Addressing

LAPB Commands and Responses - Link Level

format	command	response	encoding
information transfer	I (information)		876 5 4321 (R) p (S)0
supervisory	RR (receive ready)	RR (receive ready)	(R)P/F0001
	RNR (receive not ready)	RNR (receive not ready)	(R)P/F0101
	REJ (reject)	REJ (reject)	(R)P/F1001
unnumbered	SABM (set asynchronous balanced mode)		001 P 1111
	DISC (disconnect)		010 P 0011
		DM (disconnect mode)	000 F 1111
		UA (unnumbered acknowledgment)	011 F 0011
		FRMR (frame reject)	100 F 0111

Notes:

1. LAPB commands are also called HDLC commands
2. (R) receive sequence number, (S) send sequence number
3. (R) and (S) are 3 bits long

Figure 40. LAPB Commands and Responses

Poll/Final bit P/F

All frames contain P/F, the Poll/Final bit. In command frames, the P/F bit is referred to as the P-bit. In responses frames, it is referred to as the F-bit.

Functions of the Poll/Final bit

The poll bit set to 1 is used by DCE or DTE to solicit (poll) a response from DCE or DTE, respectively.

Appendix E. Examples of LU 6.2 SNA Transaction Programs

This appendix contains examples of transaction programs written in C language, BASIC, and COBOL. For more information on how to design these transaction programs, see Chapter 11, “Designing LU 6.2/SNA Programs.” For communications configurations for sample transaction programs, see the *4690 OS: Planning, Installation, and Configuration Guide*.

C Language TP

This section provides example information for transaction programs written in C.

Example Header File Definitions for Transaction Programs

```
1  /*****
2  /*                                     Type Definitions                               */
3  /*****
4
5  typedef long          CONV_TYPE          ;
6  typedef long          DATA_RCVD         ;
7  typedef long          DEAL_TYPE          ;
8  typedef long          ERR_DIR            ;
9  typedef long          FILL_TYPE          ;
10 typedef long          PtoR_TYPE          ;
11 typedef long          RCV_TYPE           ;
12 typedef long          RtoS_TYPE          ;
13 typedef long          RETURN_CODE         ;
14 typedef long          RTN_CTRL_TYPE      ;
15 typedef long          SEND_TYPE          ;
16 typedef long          STATUS_RCVD        ;
17 typedef long          SYNC_LEVEL         ;
18
19
20 #define TRUE           1
21 #define FALSE          0
22
23 /*****
24 /*               CPI-Communications Enumerated Constants                       */
25 /*****
26
27 #define CM_BASIC_CONVERSATION      (CONV_TYPE)      0
28 #define CM_MAPPED_CONVERSATION    (CONV_TYPE)      1
29
30 #define CM_NO_DATA_RECEIVED        (DATA_RCVD)      0
31 #define CM_DATA_RECEIVED          (DATA_RCVD)      1
32 #define CM_COMPLETE_DATA_RECEIVED (DATA_RCVD)      2
33 #define CM_INCOMPLETE_DATA_RECEIVED (DATA_RCVD)    3
34
35 #define CM_DEALLOCATE_SYNC_LEVEL  (DEAL_TYPE)      0
36 #define CM_DEALLOCATE_FLUSH      (DEAL_TYPE)      1
37 #define CM_DEALLOCATE_CONFIRM    (DEAL_TYPE)      2
38 #define CM_DEALLOCATE_ABEND      (DEAL_TYPE)      3
39
40 #define CM_RECEIVE_ERROR          (ERR_DIR)         0
41 #define CM_SEND_ERROR            (ERR_DIR)         1
42
43 #define CM_FILL_LL               (FILL_TYPE)        0
44 #define CM_FILL_BUFFER           (FILL_TYPE)        1
45
46 #define CM_PREP_TO_RECEIVE_SYNC_LEVEL (PtoR_TYPE)   0
47 #define CM_PREP_TO_RECEIVE_FLUSH    (PtoR_TYPE)   1
48 #define CM_PREP_TO_RECEIVE_CONFIRM  (PtoR_TYPE)   2
49
50 #define CM_RECEIVE_AND_WAIT       (RCV_TYPE)        0
```

C Header File Def. for TP

```

51 #define CM_RECEIVE_IMMEDIATE      (RCV_TYPE)      1
52
53 #define CM_REQ_TO_SEND_NOT_RECEIVED (RtoS_TYPE)    0
54 #define CM_REQ_TO_SEND_RECEIVED   (RtoS_TYPE)    1
55
56 #define CM_OK                      (RETURN_CODE)   0
57 #define CM_ALLOCATE_FAILURE_NO_RETRY (RETURN_CODE) 1
58 #define CM_ALLOCATE_FAILURE_RETRY  (RETURN_CODE)  2
59 #define CM_CONVERSATION_TYPE_MISMATCH (RETURN_CODE) 3
60 #define CM_SECURITY_NOT_VALID       (RETURN_CODE)  6
61 #define CM_SYNC_LVL_NOT_SUPPORTED_PGM (RETURN_CODE) 8
62 #define CM_TPN_NOT_RECOGNIZED       (RETURN_CODE)  9
63 #define CM_TP_NOT_AVAILABLE_NO_RETRY (RETURN_CODE) 10
64 #define CM_TP_NOT_AVAILABLE_RETRY   (RETURN_CODE) 11
65 #define CM_DEALLOCATED_ABEND        (RETURN_CODE) 17
66 #define CM_DEALLOCATED_NORMAL       (RETURN_CODE) 18
67 #define CM_PARAMETER_ERROR           (RETURN_CODE) 19
68 #define CM_PRODUCT_SPECIFIC_ERROR   (RETURN_CODE) 20
69 #define CM_PROGRAM_ERROR_NO_TRUNC   (RETURN_CODE) 21
70 #define CM_PROGRAM_ERROR_PURGING    (RETURN_CODE) 22
71 #define CM_PROGRAM_ERROR_TRUNC      (RETURN_CODE) 23
72 #define CM_PROGRAM_PARAMETER_CHECK   (RETURN_CODE) 24
73 #define CM_PROGRAM_STATE_CHECK       (RETURN_CODE) 25
74 #define CM_RESOURCE_FAILURE_NO_RETRY (RETURN_CODE) 26
75 #define CM_RESOURCE_FAILURE_RETRY    (RETURN_CODE) 27
76 #define CM_UNSUCCESSFUL              (RETURN_CODE) 28
77
78 #define CM_WHEN_SESSION_ALLOCATED    (RTN_CTRL_TYPE) 0
79 #define CM_IMMEDIATE                 (RTN_CTRL_TYPE) 1
80
81
82 #define CM_BUFFER_DATA              (SEND_TYPE)    0
83 #define CM_SEND_AND_FLUSH           (SEND_TYPE)    1
84 #define CM_SEND_AND_CONFIRM         (SEND_TYPE)    2
85 #define CM_SEND_AND_PREP_TO_RECEIVE (SEND_TYPE)    3
86 #define CM_SEND_AND_DEALLOCATE      (SEND_TYPE)    4
87
88 #define CM_NO_STATUS_RECEIVED        (STATUS_RCVD)  0
89 #define CM_SEND_RECEIVED             (STATUS_RCVD)  1
90 #define CM_CONFIRM_RECEIVED          (STATUS_RCVD)  2
91 #define CM_CONFIRM_SEND_RECEIVED     (STATUS_RCVD)  3
92 #define CM_CONFIRM_DEALLOC_RECEIVED  (STATUS_RCVD)  4
93
94 #define CM_NONE                     (SYNC_LEVEL)   0
95 #define CM_CONFIRM                   (SYNC_LEVEL)   1
96
97
98 /*****
99  /*      CPI-Communications FUNCTION PROTOTYPES      */
100  *****/
101
102 extern void cmaccp(char *, long *) ;
103 extern void cmallc(char *, long *) ;
104 extern void cmcfm(char *, long*, long *) ;
105 extern void cmcfmd(char *, long *) ;
106 extern void cmdeal(char *, long *) ;
107 extern void cmect(char *, long*, long *) ;
108 extern void cmemn(char *, char*, long *, long *) ;
109 extern void cmepln(char *, char *, long *, long *) ;
110 extern void cmesl(char *, long *, long *) ;
111 extern void cmflus(char *, long *) ;
112 extern void cminit(char *, char *, long *) ;
113 extern void cmptr(char *, long *) ;
114 extern void cmrcv(char *, char *, long *, long *, long *, long *, long *, long *) ;
115 extern void cmrts(char *, long *) ;
116 extern void cmsct(char *, long *, long *) ;
117 extern void cmsdt(char *, long *, long *) ;

```

```

118 extern void cmsed(char *, long *, long *) ;
119 extern void cmsend(char *, char *, long *, long *, long *) ;
120 extern void cmserr(char *, long *, long *) ;
121 extern void cmsf(char *, long *, long *) ;
122 extern void cmsld(char *, char *, long *, long *) ;
123 extern void cmsmn(char *, char *, long *, long *) ;
124 extern void cmspln(char *, char *, long *, long *) ;
125 extern void cmsptr(char *, long *, long *) ;
126 extern void cmsrc(char *, long *, long *) ;
127 extern void cmsrt(char *, long *, long *) ;
128 extern void cmsrl(char *, long *, long *) ;
129 extern void cmsst(char *, long *, long *) ;
130 extern void cmsstp(char *, char *, long *, long *) ;
131 extern void cmtrts(char *, long *, long *) ;
132
133 /*****
134  /*  NON-CPI Communications Function Prototypes 4690 Extensions  */
135  *****/
136
137 extern void xcmsto(char *, long *, long*);          /* set timeout          */
138 extern void xcmgle(long *, long *, long *, long*); /* get last error       */
139 extern void xcmel(char *, long *, long *);          /* enable link          */
140 extern void xcmcdl(char *, long *, long *);          /* cond disable link    */
141 extern void xcmdl(char *, long *, long *);          /* disable link         */
142 extern void xcmql(char *, long *, long *);          /* query link           */
143

```

Example CPI for Communications File Requester Program

```

1  /*****
2  /*
3  /*      Example File Requester Program TPR.C for 4690 OS
4  /*
5  /*  This sample file requester transaction program allocates a conversation
6  /*  with file server program TPS to transfer the file name indicated as
7  /*  a command line argument.  It can be invoked from command mode by
8  /*  issuing TPR <requested_filename>.  The following calls are
9  /*  issued in this program :
10 /*
11 /*          CMINIT (Initialize_Conversation)
12 /*          CMSSL  (Set_Sync_Level)
13 /*          CMALLC (Allocate_Conversation)
14 /*          CMSEND (Send_Data)
15 /*          CMCFM  (Confirm)
16 /*          CMRCV  (Receive_Data)
17 /*          CMCFMD (Confirmed)
18 /*
19 /*  A Symbolic Destination Name of TPR must be configured in 4690 Controller
20 /*  Communications Configuration specifying Local LU, Partner LU, MODE and
21 /*  Partner TP Name.
22 /*
23 /*  The compiled source code was linked with the 4690 CPI-Communications
24 /*  "C" link library ADXHSV2L.L86 shipped on the 4690 Optionals.
25 /*
26 /*  (C) COPYRIGHT IBM CORP. 1990
27 /*  LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
28 /*  ALL RIGHTS RESERVED
29 /*
30 *****/
31 #include <stdlib.h>
32 #include <stdio.h>
33 #include <string.h>
34 #include "cpic_c.h"
35
36
37 /*****
38 /*          CPI-Communications Parameter Declarations
39 */

```

CPI for Communications Requester TP for C

```

39  /*****
40  char          conv_id[8];          /* conversation id          */
41  char          *sym_dest_name;      /* symbolic destination    */
42  long          sync_level;          /* sync level for conversation */
43  long          rc;                  /* cpic return code        */
44
45  long          data_rcvd =          /* cmrcv data received parm */
46          CM_NO_DATA_RECEIVED;      /* initialized to no data rcvd */
47  long          status_rcvd =        /* cmrcv status received parm */
48          CM_NO_STATUS_RECEIVED;    /* initialized to no status rcvd */
49  long          requested_length = 0; /* cmrcv requested length   */
50  long          received_length = 0; /* cmrcv received length    */
51  long          req_to_send_rcvd =   /* cmsend/cmrcv rts_rcvd    */
52          CM_REQ_TO_SEND_NOT_RECEIVED;
53  long          send_length = 0;     /* SEND send data length    */
54
55
56  /*****
57  /*          File Transfer Variable Declarations          */
58  /*****
59  FILE          *file_ptr = NULL;    /* requested file name ptr  */
60  char          file_name[102];      /* requested file name      */
61  int           file_name_len;       /* requested file name length */
62  int           sys_err = FALSE;     /* system error flag        */
63
64  char          *buffer;             /* receive buffer           */
65  int           write_len;           /* amount written to file   */
66  unsigned      buf_size = 4096;     /* size of data buffer - can be */
67                                     /* up to 32K                */
68
69
70  /*****
71  /*          Program Function Prototypes          */
72  /*****
73  void disp_err(char *);             /* error message display    */
74  void receive(void);               /* issue cmrcv verb        */
75
76
77
78  /*****
79  /*          Main TPR.C Begins Here          */
80  /*****
81  void main(int argc, char *argv[])
82  {
83      if (argc == 2)                 /* check for argument      */
84      {
85          file_name_len = strlen(argv[1]); /* length of file name    */
86          strncpy(file_name, argv[1], 100); /* copy file name from args */
87      }
88      else
89      {
90          sys_err = TRUE;             /* set error flag         */
91                                     /* display error message   */
92          disp_err("Invalid/missing command-line argument(s).");
93          return;                     /* exit                    */
94      }
95                                     /* get memory for buffer   */
96      buffer = (char *)calloc(buf_size, sizeof(char));
97
98      if (buffer == NULL)             /* check for memory obtained */
99      {
100          sys_err = TRUE;             /* set error flag         */
101                                     /* display error message   */
102          disp_err("Unable to obtain memory for data buffer.");
103          return;                     /* exit                    */
104      }
105

```

```

106  /*** Setup Symbolic Destination Name - points to a configuration record ****/
107  sym_dest_name = "TPR      ";          /* sym_dest_name = 8 chars */
108
109  cminit(conv_id,                      /* initialize conversation */
110         sym_dest_name,
111         &rc);
112
113  if (rc != CM_OK)                    /* check for init errors */
114  {
115      sys_err = TRUE;                  /* set error flag */
116      /* display error message */
117      disp_err("Initialize_Conversation error.");
118      free(buffer);
119      return;                          /* exit */
120  }
121  else printf("\nInitialized Conversation\n");
122
123
124
125  /*** Set sync level for conversation to confirm ***/
126  sync_level = CM_CONFIRM;
127  cmssl(conv_id,                      /* set sync level = confirm */
128        &sync_level,                  /* must be ptr to sync_level */
129        &rc);
130
131  if (rc != CM_OK)                    /* check for set errors */
132  {
133      sys_err = TRUE;                  /* set error flag */
134      disp_err("Set_Sync_Level error."); /* display error message */
135      free(buffer);
136      return;                          /* exit */
137  }
138  else printf("\nSet sync level to confirm\n");
139
140
141
142  cmalloc(conv_id,                    /* allocate conversation */
143         &rc);
144
145  if (rc != CM_OK)                    /* check for allocate error */
146  {
147      sys_err = TRUE;                  /* set error flag */
148      disp_err("Allocate error.");      /* display error message */
149      free(buffer);
150      return;                          /* exit */
151  }
152  else printf("\nAllocated conversation\n");
153
154
155
156  send_length = file_name_len; /* set send_length to length */
157                          /* of file name */
158  cmsend(conv_id,                    /* send file name to server */
159        file_name,
160        &send_length,
161        &req_to_send_rcvd,
162        &rc);
163
164  if (rc != CM_OK)                    /* check for send_data error */
165  {
166      sys_err = TRUE;                  /* set error flag */
167      disp_err("Send_Data error ");    /* display error message */
168      printf("Return Code = %ld\n", rc);
169      free(buffer);
170      return;                          /* exit */
171  }
172  else printf("\nSent request to server\n");

```

CPI for Communications Requester TP for C

```
173
174
175
176 cmcfm(conv_id,          /* issue confirm to make sure*/
177        &req_to_send_rcvd, /* server received filename */
178        &rc);
179
180 if (rc != CM_OK)          /* check for confirm error */
181 {
182     sys_err = TRUE;      /* set error flag */
183     disp_err("Confirm error "); /* display error message */
184     printf("Return Code = %ld\n", rc );
185     free(buffer);
186     return;              /* exit */
187 }
188 else printf("\nConfirm Sent\n");
189
190
191 /*** Receive file from server as long as it is sending data ***/
192 while (TRUE)
193 {
194     receive();           /* do cmrcv to get file */
195
196     if (! sys_err)       /* check for receive error */
197     {
198         if (file_ptr == NULL) /* check for file open */
199         {
200             /* if not open then */
201             /* open file for writing */
202             file_ptr = fopen(file_name, "wb");
203
204             if (file_ptr == NULL) /* check for open error */
205             {
206                 sys_err = TRUE; /* set error flag */
207                 /* display error message */
208                 disp_err("Unable to open file.");
209                 break;          /* exit loop */
210             }
211
212             /* check for data received */
213             if ((data_rcvd == CM_COMPLETE_DATA_RECEIVED) ||
214                 (data_rcvd == CM_INCOMPLETE_DATA_RECEIVED))
215             {
216                 printf("\nReceive data\n");
217                 /* write data to file */
218                 write_len = fwrite(buffer, sizeof(char),
219                                     (int)received_length, file_ptr);
220
221                 /* check for write error */
222                 if (write_len != received_length)
223                 {
224                     sys_err = TRUE; /* set error flag */
225                     /* display error message */
226                     disp_err("Error writing to file.");
227                     break;          /* exit loop */
228                 }
229             }
230         }
231     }
232     else
233     {
234         if (data_rcvd != CM_NO_DATA_RECEIVED)
235         {
236             sys_err = TRUE; /* set error flag */
237             /* display error message */
238             disp_err("Error receiving requested file.");
239             break;          /* exit loop */
240         }
241     }
242 }
```



```

240
241                                     /* check for server complete */
242     if (status_rcvd == CM_CONFIRM_DEALLOC_RECEIVED)
243     {
244         printf("\nReceived deallocate confirm\n");
245         fclose(file_ptr); /* close file */
246
247         cmcfmd(conv_id, /* confirmed */
248               &rc);
249         printf("\nSent Confirmed\n");
250
251         break; /* exit loop */
252     }
253 }
254 else
255 {
256                                     /* receive error with file.. */
257                                     /* ..not open indicates.. */
258                                     /* ..server couldn't find.. */
259     if (file_ptr == NULL) /* ..requested file */
260     {
261         disp_err("File not found at server.");
262     }
263     else
264     {
265         disp_err("Error receiving requested file.");
266     }
267
268     break;
269 }
270 }
271
272 free(buffer); /* free buffer memory */
273
274 return;
275
276 } /* end main */
277
278
279 /*****
280  /* Display Error Message */
281  *****/
282 void disp_err(char *msg)
283 {
284     printf("\n%s\n", msg); /* display message */
285     return;
286 }
287
288 /*****
289  /* Receive Data */
290  *****/
291 void receive(void)
292 {
293     requested_length = buf_size; /* set length to receive */
294
295     cmrcv(conv_id, /* receive */
296           buffer,
297           &requested_length,
298           &data_rcvd,
299           &received_length,
300           &status_rcvd,
301           &req_to_send_rcvd,
302           &rc);
303
304     if (rc != CM_OK) /* check for receive errors */
305     {
306         sys_err = TRUE; /* set error flag */

```

CPI for Communications Requester TP for C

```
307     disp_err("Receive error.");          /* display error message */
308 }
309
310 return;
311 }
312
```

Example CPI for Communications File Server Program

```
1  /*****
2  /*
3  /* Example File Server Program TPS.C for 4690 OS
4  /*
5  /* This sample file server transaction program accepts a conversation
6  /* allocated by partner transaction program TPR. It starts
7  /* as a 4690 Background Application. The requester first sends the
8  /* file name of the requested file and this server sends it to the
9  /* requester if the file exists. An error notification is sent if the
10 /* file does not exist on this controller. The following calls
11 /* are contained in this program :
12 /*
13 /*             CMAACP (Accept Conversation)
14 /*             CMRCV  (Receive_Data)
15 /*             CMCFMD (Confirmed)
16 /*             CMSEND (Send_Data)
17 /*             CMSERR (Send_Error)
18 /*             CMSDT  (Set_Deallocate_Type)
19 /*             CMDEAL (Deallocate)
20 /*
21 /* A Remotely Attachable Local TP record must be configured in 4690
22 /* Controller Communications Configuration specifying a TP Name identical
23 /* to that configured in the TPR Symbolic Destination Name - Partner TP
24 /* Name. The conversation type is Mapped and the sync level is Confirm.
25 /* The filespec should contain the drive, subdirectory and filename of
26 /* the executable copy of this server transaction program, e.g.
27 /* C:\ADX_UPGM\TPS.286.
28 /* Also, the Remotely Attachable Local TP record name must be referenced
29 /* in a Symbolic Destination Name record on the server controller.
30 /*
31 /* Note : Because background applications cannot write directly to the
32 /* screen, all messages that this program generates are written
33 /* to a log file named TPS.LOG created in the ADX_UPGM subdirectory.
34 /*
35 /* The compiled source code was linked with the 4690 CPI-Communications
36 /* "C" link library ADXHSV2L.L86 shipped on the 4690 Optionals.
37 /*
38 /* (C) COPYRIGHT IBM CORP. 1990
39 /* LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
40 /* ALL RIGHTS RESERVED
41 /*
42 /*****/
43 #include <stdlib.h>
44 #include <stdio.h>
45 #include <string.h>
46 #include "cpic_c.h"
47
48
49 /*****/
50 /*      CPI-Communications Parameter Declarations
51 /*****/
52 char      conv_id[8];          /* conversation id
53 long      deal_type;           /* deallocate type
54 long      rc;                  /* cpic return code
55
56 long      data_rcvd =          /* cmrcv data received parm
57          CM_NO_DATA_RECEIVED;  /* initialized to no data rcvd
58 long      status_rcvd =       /* cmrcv status received parm
59          CM_NO_STATUS_RECEIVED; /* initialized to no status rcvd*/
```

```

60 long      requested_length = 0;      /* cmrcv requested length */
61 long      received_length = 0;      /* cmrcv received length */
62 long      req_to_send_rcvd =        /* cmsend/cmrcv rts_rcvd */
63          CM_REQ_TO_SEND_NOT_RECEIVED;
64 long      send_length = 0;          /* SEND send data length */
65
66
67 /*****
68 /*          File Transfer Variable Declarations          */
69 *****/
70 FILE      *file_ptr = NULL;          /*
requested file name ptr */
71 FILE      *log_file = NULL;          /* log file pointer */
72 char      *log_fname="ADX_UPGM:TPS.LOG"; /* log file name */
73 char      *log_fmode="w";            /* write */
74 int       sys_err = FALSE;           /* system error flag */
75
76 char      *buffer;                  /* receive buffer */
77 unsigned   buf_size = 4096;          /* size of data buffer */
78
79
80 /*****
81 /*          Program Function Prototypes          */
82 *****/
83 void disp_err(char *);               /* error message display */
84 void receive(void);                  /* issue cmrcv verb */
85 void send_data(void);                /* issue cmsend verb */
86 void send_error(void);               /* issue cmserr verb */
87
88
89
90 /*****
91 /*          Main TPS.C Begins Here          */
92 *****/
93 void main()
94 {
95     /* Open log file for message logging ... if cannot open then exit */
96     if ((log_file = fopen( log_fname,
log_fmode )) == NULL) return;
97
98     /* get memory for buffer */
99     buffer = (char *)calloc(buf_size, sizeof(char));
100
101     if (buffer == NULL)                /* check for memory obtained */
102     {
103         sys_err = TRUE;                /* set error flag */
104         /* display error message */
105         disp_err("Unable to obtain memory for data buffer.");
106         return;                        /* exit */
107     }
108
109     cmaccp(conv_id,                    /* accept conversation */
110            &rc);
111
112     if (rc != CM_OK)                   /* check for accept errors */
113     {
114         sys_err = TRUE;                /* set error flag */
115         disp_err("Accept_Conversation error."); /* display error message */
116         return;                        /* exit */
117     }
118     receive();                          /* wait for a file request */
119
120     if (! sys_err)                     /* check for previous error */
121     {
122         /* check for complete data */
123         if (data_rcvd == CM_COMPLETE_DATA_RECEIVED) /* of filename */

```

```

124     {
125         receive();                                /* wait for confirm status */
126
127         if (! sys_err)                            /* check for previous error */
128         {
129             /* check for send permission */
130             receive();                            /* wait for send permission */
131             if (status_rcvd == CM_SEND_RECEIVED)
132             {
133                 /* open file for reading */
134                 file_ptr = fopen(buffer, "rb");
135             }
136             else
137             {
138                 sys_err = TRUE;                    /* set error flag */
139                 /* display error message */
140                 disp_err("Permission to send not received.");
141             }
142         }
143     }
144     else
145     {
146         sys_err = TRUE;                            /* set error flag */
147         /* display error message */
148         disp_err("Complete request not received.");
149     }
150 }
151
152 if (file_ptr != NULL)                            /* check for open error */
153 {
154     do
155     {
156         /* read from file */
157         send_length = fread(buffer, sizeof(char), buf_size, file_ptr);
158
159         if (send_length < buf_size)                /* check for partial buffer */
160         {
161             if (ferror(file_ptr))                  /* check for file read error */
162             {
163                 sys_err = TRUE;                    /* set error flag */
164                 send_error();                      /* notify requester of error */
165                 /* display error message */
166                 disp_err("Error reading requested file.");
167                 break;                             /* exit loop */
168             }
169         }
170
171         send_data();                                /* send file to requester */
172     }
173     while ((!! feof(file_ptr)) &&                  /* loop until e-o-f or error */
174            (! sys_err));
175
176     fclose(file_ptr);                             /* close file */
177 }
178 else
179 {
180     sys_err = TRUE;                                /* set error flag */
181     send_error();                                  /* notify requester of error */
182     /* display error message */
183     disp_err("Unable to open requested file.");
184
185     deal_type = CM_DEALLOCATE_ABEND;
186     cmsdt(conv_id,                                /* set deal_type = abend */
187           &deal_type,
188           &rc);
189 }
190

```

```

191  cmdeal(conv_id,                      /* deallocate conversation */
192         &rc);
193
194  free(buffer);                        /* free buffer memory */
195
196  return;
197
198 } /* end main */
199
200
201
202 /*****
203  /*          Disp_err - Writes a message to log file          */
204  *****/
205 void disp_err(char *msg)
206 {
207     fprintf( log_file, "%s\n", msg );
208     return;
209 }
210
211 /*****
212  /*          Receive Executes CMRCV          */
213  *****/
214 void receive(void)
215 {
216     requested_length = buf_size;      /* set length to receive */
217
218     cmrcv(conv_id,                    /* receive */
219           buffer,
220           &requested_length,
221           &data_rcvd,
222           &received_length,
223           &status_rcvd,
224           &req_to_send_rcvd,
225           &rc);
226
227
228     if (rc != CM_OK)                  /* check for receive errors */
229     {
230         sys_err = TRUE;                /* set error flag */
231         disp_err("Receive error.");    /* display error message */
232     }
233
234     if (status_rcvd == CM_CONFIRM_RECEIVED)
235     {
236         fprintf( log_file, "Issuing CONFIRMED\n" );
237         cmcfmd( conv_id,
238                 &rc );
239     }
240
241
242     return;
243 }
244
245 /*****
246  /*          Send_Error Executes CMSERR          */
247  *****/
248 void send_error(void)
249 {
250     cmserr(conv_id,                  /* send error */
251           &req_to_send_rcvd,
252           &rc);
253
254     if (rc != CM_OK)                  /* check for send_error error*/
255     {
256         sys_err = TRUE;                /* set error flag */
257         disp_err("Send_Error error."); /* display error message */

```

CPI for Communications Server TP for C

```
258     }
259
260     return;
261 }
262
263 /*****
264  /*          Send_Data Executes CMSEND          */
265  *****/
266 void send_data(void)
267 {
268     cmsend(conv_id,          /* send data          */
269           buffer,
270           &send_length,
271           &req_to_send_rcvd,
272           &rc);
273
274     if (rc != CM_OK)          /* check for send_data error */
275     {
276         sys_err = TRUE;      /* set error flag          */
277         disp_err("Send_Data error."); /* display error message */
278     }
279
280     return;
281 }
282
```

BASIC TPs

This section provides example information for transaction programs written in BASIC.

Variable Type for 4680 BASIC Programs

```
! *****/
!          CPI-Communications Enumerated Constants          */
! *****/
```

```
INTEGER*4 CM.BASIC.CONVERSATION
INTEGER*4 CM.MAPPED.CONVERSATION
```

```
INTEGER*4 CM.NO.DATA.RECEIVED
INTEGER*4 CM.DATA.RECEIVED
INTEGER*4 CM.COMPLETE.DATA.RECEIVED
INTEGER*4 CM.INCOMPLETE.DATA.RECEIVED
```

```
INTEGER*4 CM.DEALLOCATE.SYNC.LEVEL
INTEGER*4 CM.DEALLOCATE.FLUSH
INTEGER*4 CM.DEALLOCATE.CONFIRM
INTEGER*4 CM.DEALLOCATE.ABEND
```

```
INTEGER*4 CM.RECEIVE.ERROR
INTEGER*4 CM.SEND.ERROR
```

```
INTEGER*4 CM.FILL.LL
INTEGER*4 CM.FILL.BUFFER
```

```
INTEGER*4 CM.PREP.TO.RECEIVE.SYNC.LEVEL
INTEGER*4 CM.PREP.TO.RECEIVE.FLUSH
INTEGER*4 CM.PREP.TO.RECEIVE.CONFIRM
```

```
INTEGER*4 CM.RECEIVE.AND.WAIT
INTEGER*4 CM.RECEIVE.IMMEDIATE
```

```
INTEGER*4 CM.REQ.TO.SEND.NOT.RECEIVED
INTEGER*4 CM.REQ.TO.SEND.RECEIVED
```

```
INTEGER*4 CM.OK
INTEGER*4 CM.ALLOCATE.FAILURE.NO.RETRY
```

```

INTEGER*4 CM.ALLOCATE.FAILURE.RETRY
INTEGER*4 CM.CONVERSATION.TYPE.MISMATCH
INTEGER*4 CM.SECURITY.NOT.VALID
INTEGER*4 CM.SYNC.LVL.NOT.SUPPORTED.PGM
INTEGER*4 CM.TPN.NOT.RECOGNIZED
INTEGER*4 CM.TP.NOT.AVAILABLE.NO.RETRY
INTEGER*4 CM.TP.NOT.AVAILABLE.RETRY
INTEGER*4 CM.DEALLOCATED.ABEND
INTEGER*4 CM.DEALLOCATED.NORMAL
INTEGER*4 CM.PARAMETER.ERROR
INTEGER*4 CM.PRODUCT.SPECIFIC.ERROR
INTEGER*4 CM.PROGRAM.ERROR.NO.TRUNC
INTEGER*4 CM.PROGRAM.ERROR.PURGING
INTEGER*4 CM.PROGRAM.ERROR.TRUNC
INTEGER*4 CM.PROGRAM.PARAMETER.CHECK
INTEGER*4 CM.PROGRAM.STATE.CHECK
INTEGER*4 CM.RESOURCE.FAILURE.NO.RETRY
INTEGER*4 CM.RESOURCE.FAILURE.RETRY
INTEGER*4 CM.UNSUCCESSFUL

```

```

INTEGER*4 CM.WHEN.SESSION.ALLOCATED
INTEGER*4 CM.IMMEDIATE

```

```

INTEGER*4 CM.BUFFER.DATA
INTEGER*4 CM.SEND.AND.FLUSH
INTEGER*4 CM.SEND.AND.CONFIRM
INTEGER*4 CM.SEND.AND.PREP.TO.RECEIVE
INTEGER*4 CM.SEND.AND.DEALLOCATE

```

```

INTEGER*4 CM.NO.STATUS.RECEIVED
INTEGER*4 CM.SEND.RECEIVED
INTEGER*4 CM.CONFIRM.RECEIVED
INTEGER*4 CM.CONFIRM.SEND.RECEIVED
INTEGER*4 CM.CONFIRM.DEALLOC.RECEIVED

```

```

INTEGER*4 CM.NONE
INTEGER*4 CM.CONFIRM

```

```

!
! Conversation Characteristics Variable Declarations
!
INTEGER*4 CONVERSATION.TYPE
INTEGER*4 DEALLOCATE.TYPE
INTEGER*4 ERROR.DIRECTION
INTEGER*4 PARM.FILL
STRING   LOG.DATA
INTEGER*4 LOG.DATA.LENGTH
STRING   MODE.NAME
INTEGER*4 MODE.NAME.LENGTH
STRING   PARTNER.LU.NAME
INTEGER*4 PARTNER.LU.NAME.LENGTH
INTEGER*4 PREPARE.TO.RECEIVE.TYPE
INTEGER*4 RECEIVE.TYPE
INTEGER*4 RETURN.CONTROL
INTEGER*4 SEND.TYPE
INTEGER*4 SYNC.LEVEL
STRING   TP.NAME
INTEGER*4 TP.NAME.LENGTH
!
! CPI-Communications Extension Function Variable Declarations
!
STRING   LINK.NAME
INTEGER*4 CPIC.VERB
INTEGER*4 CPIC.RC
INTEGER*4 ERR.PARM1, ERR.PARM2

```

Variable Type for BASIC TP

```
INTEGER*4 TIMEOUT.VAL
!  
! CPI-Communications Parameter Declarations  
!  
STRING    SEND.BUFFER  
STRING    RECEIVE.BUFFER  
STRING    SYM.DEST.NAME  
STRING    CONVERSATION.ID  
INTEGER*4 REQUESTED.LENGTH  
INTEGER*4 DATA.RECEIVED  
INTEGER*4 RECEIVED.LENGTH  
INTEGER*4 STATUS.RECEIVED  
INTEGER*4 REQUEST.TO.SEND.RECEIVED  
INTEGER*4 RETURN.CODE  
INTEGER*4 SEND.LENGTH  
!
```

CPI for Communications Subroutine Definitions for 4680 BASIC

```
! *****/  
!      4680 BASIC CPI-Communications External Function Declarations      */  
! *****/
```

```
SUB CMACCP(CONVERSATION.ID, RETURN.CODE) EXTERNAL  
STRING CONVERSATION.ID  
INTEGER*4 RETURN.CODE  
END SUB
```

```
SUB CMALLC(CONVERSATION.ID, RETURN.CODE) EXTERNAL  
STRING CONVERSATION.ID  
INTEGER*4 RETURN.CODE  
END SUB
```

```
SUB CMCFM(CONVERSATION.ID, REQUEST.TO.SEND.RECEIVED, \  
RETURN.CODE) EXTERNAL  
STRING CONVERSATION.ID  
INTEGER*4 REQUEST.TO.SEND.RECEIVED  
INTEGER*4 RETURN.CODE  
END SUB
```

```
SUB CMCFMD(CONVERSATION.ID, RETURN.CODE) EXTERNAL  
STRING CONVERSATION.ID  
INTEGER*4 RETURN.CODE  
END SUB
```

```
SUB CMDEAL(CONVERSATION.ID, RETURN.CODE) EXTERNAL  
STRING CONVERSATION.ID  
INTEGER*4 RETURN.CODE  
END SUB
```

```
SUB CMFLUS(CONVERSATION.ID, RETURN.CODE) EXTERNAL  
STRING CONVERSATION.ID  
INTEGER*4 RETURN.CODE  
END SUB
```

```
SUB CMINIT(CONVERSATION.ID, SYM.DEST.NAME, RETURN.CODE) EXTERNAL  
STRING SYM.DEST.NAME, CONVERSATION.ID  
INTEGER*4 RETURN.CODE  
END SUB
```

```
SUB CMPTR(CONVERSATION.ID, RETURN.CODE) EXTERNAL  
STRING CONVERSATION.ID  
INTEGER*4 RETURN.CODE  
END SUB
```

```
SUB CMRCV(CONVERSATION.ID, BUFFER, REQUESTED.LENGTH, DATA.RECEIVED, \  
RECEIVED.LENGTH, STATUS.RECEIVED, REQUEST.TO.SEND.RECEIVED, \  
RETURN.CODE) EXTERNAL
```



```

STRING CONVERSATION.ID, BUFFER
INTEGER*4 REQUESTED.LENGTH, DATA.RECEIVED, RECEIVED.LENGTH
INTEGER*4 STATUS.RECEIVED, REQUEST.TO.SEND.RECEIVED, RETURN.CODE
END SUB

SUB CMRTS(CONVERSATION.ID, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 RETURN.CODE
END SUB

SUB CMSEND(CONVERSATION.ID, BUFFER, SEND.LENGTH, \
           REQUEST.TO.SEND.RECEIVED, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID, BUFFER
INTEGER*4 SEND.LENGTH, REQUEST.TO.SEND.RECEIVED, RETURN.CODE
END SUB

SUB CMSERR(CONVERSATION.ID, REQUEST.TO.SEND.RECEIVED, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 REQUEST.TO.SEND.RECEIVED, RETURN.CODE
END SUB

SUB CMTRTS(CONVERSATION.ID, REQUEST.TO.SEND.RECEIVED, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 REQUEST.TO.SEND.RECEIVED, RETURN.CODE
END SUB

SUB CMECT(CONVERSATION.ID, CONVERSATION.TYPE, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 CONVERSATION.TYPE, RETURN.CODE
END SUB

SUB CMEMN(CONVERSATION.ID, MODE.NAME, MOSE.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID, MODE.NAME
INTEGER*4 MODE.NAME.LENGTH, RETURN.CODE
END SUB

SUB CMEPLN(CONVERSATION.ID, PARTNER.LU.NAME, \
           PARTNER.LU.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID, PARTNER.LU.NAME
INTEGER*4 PARTNER.LU.NAME.LENGTH, RETURN.CODE
END SUB

SUB CMESL(CONVERSATION.ID, SYNC.LEVEL, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 SYNC.LEVEL, RETURN.CODE
END SUB

SUB CMSCT(CONVERSATION.ID, CONVERSATION.TYPE, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 CONVERSATION.TYPE, RETURN.CODE
END SUB

SUB CMSDT(CONVERSATION.ID, DEALLOCATE.TYPE, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 DEALLOCATE.TYPE, RETURN.CODE
END SUB

SUB CMSED(CONVERSATION.ID, ERROR.DIRECTION, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 ERROR.DIRECTION, RETURN.CODE
END SUB

SUB CMSF(CONVERSATION.ID, PARM.FILL, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 PARM.FILL, RETURN.CODE
END SUB

```

CPI for Communications Subroutine Defs. for BASIC TP

```
SUB CMSLD(CONVERSATION.ID, LOG.DATA, LOG.DATA.LENGTH, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID, LOG.DATA
INTEGER*4 LOG.DATA.LENGTH, RETURN.CODE
END SUB
```

```
SUB CMSMN(CONVERSATION.ID, MODE.NAME, MODE.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID, MODE.NAME
INTEGER*4 MODE.NAME.LENGTH, RETURN.CODE
END SUB
```

```
SUB CMSPLN(CONVERSATION.ID, PARTNER.LU.NAME, PARTNER.LU.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID, PARTNER.LU.NAME
INTEGER*4 PARTNER.LU.NAME.LENGTH, RETURN.CODE
END SUB
```

```
SUB CMSPTR(CONVERSATION.ID, PREPARE.TO.RECEIVE.TYPE, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 PREPARE.TO.RECEIVE.TYPE, RETURN.CODE
END SUB
```

```
SUB CMSRT(CONVERSATION.ID, RECEIVE.TYPE, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 RECEIVE.TYPE, RETURN.CODE
END SUB
```

```
SUB CMSRC(CONVERSATION.ID, RETURN.CONTROL, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 RETURN.CONTROL, RETURN.CODE
END SUB
```

```
SUB CMSST(CONVERSATION.ID, SEND.TYPE, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 SEND.TYPE, RETURN.CODE
END SUB
```

```
SUB CMSSL(CONVERSATION.ID, SYNC.LEVEL, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 SYNC.LEVEL, RETURN.CODE
END SUB
```

```
SUB CMSTPN(CONVERSATION.ID, TP.NAME, TP.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID, TP.NAME
INTEGER*4 TP.NAME.LENGTH, RETURN.CODE
END SUB
```

```
SUB XCML(LINK.NAME, LINK.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING LINK.NAME
INTEGER*4 LINK.NAME.LENGTH, RETURN.CODE
END SUB
```

```
SUB XCMCDL(LINK.NAME, LINK.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING LINK.NAME
INTEGER*4 LINK.NAME.LENGTH, RETURN.CODE
END SUB
```

```
SUB XCMDL(LINK.NAME, LINK.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING LINK.NAME
INTEGER*4 LINK.NAME.LENGTH, RETURN.CODE
END SUB
```

```
SUB XCMQL(LINK.NAME, LINK.NAME.LENGTH, RETURN.CODE) EXTERNAL
STRING LINK.NAME
INTEGER*4 LINK.NAME.LENGTH, RETURN.CODE
END SUB
```

```
SUB XCMGLE(CPIC.VERB, CPIC.RC, ERR.PARM1, ERR.PARM2) EXTERNAL
INTEGER *4 CPIC.VERB
```

```

INTEGER *4 CPIC.RC
INTEGER *4 ERR.PARM1, ERR.PARM2
END SUB

```

```

SUB XCMSTO(CONVERSATION.ID, TIMEOUT.VAL, RETURN.CODE) EXTERNAL
STRING CONVERSATION.ID
INTEGER*4 TIMEOUT.VAL, RETURN.CODE
END SUB

```

4680 BASIC Pseudonym Equates for TP

```

! *****/
!           Assign CPI-Communications Pseudonym Values          */
! *****/

```

```

CM.BASIC.CONVERSATION      = 0000H
CM.MAPPED.CONVERSATION     = 0001H

```

```

CM.NO.DATA.RECEIVED        = 0000H
CM.DATA.RECEIVED           = 0001H
CM.COMPLETE.DATA.RECEIVED  = 0002H
CM.INCOMPLETE.DATA.RECEIVED = 0003H

```

```

CM.DEALLOCATE.SYNC.LEVEL   = 0000H
CM.DEALLOCATE.FLUSH        = 0001H
CM.DEALLOCATE.CONFIRM      = 0002H
CM.DEALLOCATE.ABEND        = 0003H

```

```

CM.RECEIVE.ERROR           = 0000H
CM.SEND.ERROR              = 0001H

```

```

CM.FILL.LL                 = 0000H
CM.FILL.BUFFER             = 0001H

```

```

CM.PREP.TO.RECEIVE.SYNC.LEVEL = 0000H
CM.PREP.TO.RECEIVE.FLUSH      = 0001H
CM.PREP.TO.RECEIVE.CONFIRM    = 0002H

```

```

CM.RECEIVE.AND.WAIT        = 0000H
CM.RECEIVE.IMMEDIATE       = 0001H

```

```

CM.REQ.TO.SEND.NOT.RECEIVED = 0000H
CM.REQ.TO.SEND.RECEIVED     = 0001H

```

```

CM.OK                      = 0000H
CM.ALLOCATE.FAILURE.NO.RETRY = 0001H
CM.ALLOCATE.FAILURE.RETRY   = 0002H
CM.CONVERSATION.TYPE.MISMATCH = 0003H
CM.SECURITY.NOT.VALID      = 0006H
CM.SYNC.LVL.NOT.SUPPORTED.PGM = 0008H
CM.TPN.NOT.RECOGNIZED      = 0009H
CM.TP.NOT.AVAILABLE.NO.RETRY = 000AH
CM.TP.NOT.AVAILABLE.RETRY  = 000BH
CM.DEALLOCATED.ABEND       = 0011H
CM.DEALLOCATED.NORMAL      = 0012H
CM.PARAMETER.ERROR         = 0013H
CM.PRODUCT.SPECIFIC.ERROR  = 0014H
CM.PROGRAM.ERROR.NO.TRUNC  = 0015H
CM.PROGRAM.ERROR.PURGING   = 0016H
CM.PROGRAM.ERROR.TRUNC     = 0017H
CM.PROGRAM.PARAMETER.CHECK = 0018H
CM.PROGRAM.STATE.CHECK     = 0019H
CM.RESOURCE.FAILURE.NO.RETRY = 001AH
CM.RESOURCE.FAILURE.RETRY   = 001BH
CM.UNSUCCESSFUL            = 001CH

```

BASIC Pseudonym Equates for TP

CM.WHEN.SESSION.ALLOCATED	= 0000H
CM.IMMEDIATE	= 0001H
CM.BUFFER.DATA	= 0000H
CM.SEND.AND.FLUSH	= 0001H
CM.SEND.AND.CONFIRM	= 0002H
CM.SEND.AND.PREP.TO.RECEIVE	= 0003H
CM.SEND.AND.DEALLOCATE	= 0004H
CM.NO.STATUS.RECEIVED	= 0000H
CM.SEND.RECEIVED	= 0001H
CM.CONFIRM.RECEIVED	= 0002H
CM.CONFIRM.SEND.RECEIVED	= 0003H
CM.CONFIRM.DEALLOC.RECEIVED	= 0004H
CM.NONE	= 0000H
CM.CONFIRM	= 0001H

Example File Requester TP Program in 4680 BASIC

```
!*****
!*
!*      Example File Requester Program TPRCBAS.BAS for 4690 OS
!*
!* This sample file requester transaction program allocates a conversation
!* with file server program TPSCBAS to transfer the file name indicated as
!* a command line argument. It can be invoked from command mode by
!* issuing TPRCBAS <requested_filename>. The following
!* CPI-Communications calls issued in this program :
!*
!*      CMINIT (Initialize_Conversation)
!*      CMSSL  (Set_Sync_Level)
!*      CMALLC (Allocate_Conversation)
!*      CMSEND (Send_Data)
!*      CMCFM  (Confirm)
!*      CMRCV  (Receive_Data)
!*      CMCFMD (Confirmed)
!*
!* A Symbolic Destination Name of TPRCBAS must be configured in 4690 Ctlr
!* Communications Configuration specifying Local LU, Partner LU, MODE and
!* Partner TP Name.
!*
!* The compiled source code was linked with the 4690 CPI-Communications
!* "BASIC" library ADXHSV0L.L86 shipped on the 4690 Optionals.
!*
!* (C) COPYRIGHT IBM CORP. 1990
!* LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
!* ALL RIGHTS RESERVED
!*
!*****
!
!
%INCLUDE CPIC_BAS.DEF      ! Include Pseudonym Definitions
!
!
! Program Declarations
!
INTEGER*2 TRUE
INTEGER*2 FALSE
!
STRING BUFFER
INTEGER*2 SYS.ERR
STRING FILE.NAME
INTEGER*2 FILE.NAME.LEN
STRING WRITE.LEN
INTEGER*2 BUF.SIZE
INTEGER*2 CONTINUE
STRING ERR.MSG
```

```

STRING ERR.RC$
INTEGER*2 FILE.OPEN
INTEGER*2 I%
!
! CPI-Communications External Function Subroutines Include
!
%INCLUDE CPIC_BAS.SUB
!
! Local Program Subroutine Declarations
!
SUB DISP.MSG(MSG)
  STRING MSG
  PRINT
  PRINT MSG
END SUB
!
SUB RECEIVE
  REQUESTED.LENGTH = BUF.SIZE
  !
  CALL DISP.MSG("Receive")
  CALL CMRCV(CONVERSATION.ID,          \
             BUFFER,                   \
             REQUESTED.LENGTH,         \
             DATA.RECEIVED,           \
             RECEIVED.LENGTH,          \
             STATUS.RECEIVED,          \
             REQUEST.TO.SEND.RECEIVED, \
             RETURN.CODE)
  !
  IF RETURN.CODE <> CM.OK THEN \
    BEGIN
      SYS.ERR = TRUE
      PRINT "Receive Error rc = ", RETURN.CODE
      ERR.MSG = "Receive error."
      CALL DISP.MSG(ERR.MSG)
    ENDIF
END SUB
!
FUNCTION D2X$(LONG.DEC)
  STRING D2X$
  INTEGER*2 J
  INTEGER*4 TEMPC, TEMPD, LONG.DEC
  STRING HEXTAB, ERRNC

  HEXTAB = "0123456789ABCDEF"

  TEMPC = LONG.DEC
  ERRNC = ""
  FOR J = 1 TO 8
    TEMPD = TEMPC AND 0FH
    ERRNC = MID$(HEXTAB,(TEMPD + 1),1) + ERRNC
    TEMPC = SHIFT(TEMPC,4)
  NEXT J
  D2X$ = ERRNC
END FUNCTION
!
! Include CPI-Communications Pseudonym Equates
!
%INCLUDE CPIC_BAS.EQU
!
TRUE = 1
FALSE = 0
BUF.SIZE = 4096
SYS.ERR = FALSE
FILE.OPEN = FALSE
!
! Main Program Begins Here

```

Requester TP for BASIC

```
!
ON ERROR GOTO ERROR.HANDLER
!
FILE.NAME = COMMAND$
!
!
PRINT "Requested file = ",FILE.NAME
!
IF FILE.NAME = "" THEN \
    BEGIN
    SYS.ERR = TRUE
    ERR.MSG = "Invalid/missing command-line argument(s).\"
    CALL DISP.MSG(ERR.MSG)
    STOP
    ENDIF
!
!   Initialize Conversation
!
CALL DISP.MSG("Issuing Initialize Conversation")
SYM.DEST.NAME = "TPRCBAS "           ! Eight character sym_dest_name
CALL CMINIT(CONVERSATION.ID, \
            SYM.DEST.NAME, \
            RETURN.CODE)
!
IF RETURN.CODE <> CM.OK THEN \
    BEGIN
    SYS.ERR = TRUE
    ERR.MSG = "Initialize.Conversation error.\"
    CALL DISP.MSG(ERR.MSG)
    STOP
    ENDIF
!
!   Set sync level to Confirm
!
CALL DISP.MSG("Set Sync Level to Confirm")
SYNC.LEVEL = CM.CONFIRM
CALL CMSSL(CONVERSATION.ID, \
           SYNC.LEVEL, \
           RETURN.CODE)
!
IF RETURN.CODE <> CM.OK THEN \
    BEGIN
    SYS.ERR = TRUE
    ERR.MSG = "Set.Sync.Level error.\"
    CALL DISP.MSG(ERR.MSG)
    STOP
    ENDIF
!
!   Allocate Conversation
!
CALL DISP.MSG("Allocate Conversation")
CALL CMALLC(CONVERSATION.ID, \
            RETURN.CODE)
!
IF RETURN.CODE <> CM.OK THEN \
    BEGIN
    SYS.ERR = TRUE
    ERR.MSG = "Allocate error.\"
    CALL DISP.MSG(ERR.MSG)
    STOP
    ENDIF
!
SEND.LENGTH = LEN(FILE.NAME)
BUFFER = FILE.NAME
!
CALL DISP.MSG("Send Requested File Name to Server")
CALL CMSEND(CONVERSATION.ID, \
```

```

        BUFFER,          \
        SEND.LENGTH,     \
        REQUEST.TO.SEND.RECEIVED, \
        RETURN.CODE)
!
IF RETURN.CODE <> CM.OK THEN \
BEGIN
  SYS.ERR = TRUE
  ERR.MSG = "Send.Data error."
  CALL DISP.MSG(ERR.MSG)
  STOP
ENDIF
!
BUFFER = STRING$(BUF.SIZE, " ")
!
CONTINUE = TRUE
WHILE CONTINUE = TRUE
  CALL RECEIVE
  !
  IF NOT SYS.ERR THEN \
  BEGIN
    IF NOT FILE.OPEN THEN \
    BEGIN
      PRINT "About to create file"
      CREATE FILE.NAME RECL BUF.SIZE AS 1 BUFFSIZE BUF.SIZE
      FILE.OPEN = TRUE
    ENDIF
    !
    IF ((DATA.RECEIVED = CM.COMPLETE.DATA.RECEIVED) OR \
        (DATA.RECEIVED = CM.INCOMPLETE.DATA.RECEIVED)) THEN \
    BEGIN
      IF RECEIVED.LENGTH = BUF.SIZE THEN \
      BEGIN
        WRITE FORM "C4096"; #1; BUFFER          ! Receive 4K blocks of data
      ENDIF \
      ELSE \                                     ! Else if < 4K sent then
      BEGIN                                       ! write on byte at a time
        FOR I% = 1 TO RECEIVED.LENGTH
          PUT 1, ASC(MID$(BUFFER, I%, 1))
        NEXT I%
      ENDIF \
    ENDIF \
  ELSE \
  BEGIN
    IF DATA.RECEIVED <> CM.NO.DATA.RECEIVED THEN \
    BEGIN
      SYS.ERR = TRUE
      ERR.MSG = "Error receiving requested file."
      CALL DISP.MSG(ERR.MSG)
      CONTINUE = FALSE
    ENDIF
  ENDIF
  !
  IF CONTINUE = TRUE THEN \
  BEGIN
    IF STATUS.RECEIVED = CM.CONFIRM.DEALLOC.RECEIVED THEN \
    BEGIN
      CLOSE 1
      !
      CALL DISP.MSG("Issuing Confirmed")
      CALL CMCMD(CONVERSATION.ID, \
                  RETURN.CODE)
      !
      CONTINUE = FALSE
    ENDIF
  ENDIF
ENDIF \

```

Requester TP for BASIC

```
ELSE \
  BEGIN
  IF FILE.OPEN = FALSE THEN \
    BEGIN
    ERR.MSG = "File not found at server."
    ENDIF \
  ELSE \
    BEGIN
    ERR.MSG = "Error receiving requested file."
    ENDIF
  CONTINUE = FALSE
  ENDIF
WEND
!
STOP
!
ERROR.HANDLER:
  ERR.RC$ = D2X$(ERRN)
  CALL DISP.MSG(ERR)
  CALL DISP.MSG(ERR.RC$)
  IF ERR = "OE" THEN \
    BEGIN
    FILE.OPEN = FALSE
    SYS.ERR = TRUE
    ERR.MSG = "Unable to open file."
    CALL DISP.MSG(ERR.MSG)
    RESUME
    ENDIF
  !
  STOP
!
END
```

Example File Server TP Program in 4680 BASIC

```
!*****
!*
!* Example File Server Program TPSCBAS.BAS for 4690 OS
!*
!* This sample file server transaction program accepts a conversation
!* allocated by partner transaction program TPRCBAS. It starts
!* as a 4690 Background Application. The requester first sends the
!* file name of the requested file and this server sends it to the
!* requester if the file exists. An error notification is sent if the
!* file does not exist on this controller. The following calls
!* are contained in this program :
!*
!*          CMACCP (Accept Conversation)
!*          CMRCV  (Receive_Data)
!*          CMCFMD (Confirmed)
!*          CMSEND (Send_Data)
!*          CMSERR (Send_Error)
!*          CMSDT  (Set_Deallocate_Type)
!*          CMDEAL (Deallocate)
!*
!* A Remotely Attachable Local TP record must be configured in 4690
!* Controller Communications Configuration specifying a TP Name identical
!* to that configured in the TPR Symbolic Destination Name - Partner TP
!* Name. The conversation type is Mapped and the sync level is Confirm.
!* The filespec should contain the drive, subdirectory and filename of
!* the executable copy of this server transaction program, e.g.
!* C:\ADX_UPGM\TPSCBAS.286
!* Also, the Remotely Attachable Local TP record name must be referenced
!* in a Symbolic Destination Name record on the server controller.
!*
!* Note : Because background applications cannot write directly to the
!* screen, all messages that this program generates are written
!* to a log file named TPSCBAS.LOG created in ADX_UPGM.
```



```

!*
!* The compiled source code was linked with the 4690 CPI-Communications
!* "BASIC" library ADXHSV0L.L86 shipped on the 4690 Optionals.
!*
!* (C) COPYRIGHT IBM CORP. 1990
!* LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
!* ALL RIGHTS RESERVED
!*
!*****
!
!
%INCLUDE CPIC_BAS.DEF                                ! Include Pseudonym Definitions
!
!
! Program Declarations
!
INTEGER*2 TRUE
INTEGER*2 FALSE
!
STRING BUFFER
INTEGER*2 SYS.ERR
STRING FILE.NAME
INTEGER*2 FILE.NAME.LEN
INTEGER*2 I%, BYTE%
INTEGER*4 FILE.SIZE
INTEGER*2 NUM.OF.4K.BLOCKS
INTEGER*2 LAST.BLOCK
STRING LAST.BLOCK$
INTEGER*2 BUF.SIZE
INTEGER*2 CONTINUE
STRING ARGS
STRING ERR.MSG
INTEGER*2 FILE.OPEN
INTEGER*4 RECORD.NUM
STRING READ.ERROR$
!
! CPI-Communications External Function Subroutines Include
!
%INCLUDE CPIC_BAS.SUB
!
! Local Program Subroutine Declarations
!
SUB DISP.MSG(MSG)
    STRING MSG
    PRINT #2; MSG
END SUB
!
SUB RECEIVE
    REQUESTED.LENGTH = BUF.SIZE
    !
    CALL DISP.MSG("Receive")
    CALL CMRCV(CONVERSATION.ID, \
                BUFFER, \
                REQUESTED.LENGTH, \
                DATA.RECEIVED, \
                RECEIVED.LENGTH, \
                STATUS.RECEIVED, \
                REQUEST.TO.SEND.RECEIVED, \
                RETURN.CODE)
    !
    IF RETURN.CODE <> CM.OK THEN \
        BEGIN
            SYS.ERR = TRUE
            ERR.MSG = "Receive error."
            CALL DISP.MSG(ERR.MSG)
        ENDIF
END SUB

```

Server TP for BASIC

```
!
SUB SEND.ERROR
  CALL CMSERR(CONVERSATION.ID,      \
              REQUEST.TO.SEND.RECEIVED, \
              RETURN.CODE)
!
  IF RETURN.CODE <> CM.OK THEN \
    BEGIN
      SYS.ERR = TRUE
      ERR.MSG = "Send.Error error."
      CALL DISP.MSG(ERR.MSG)
    ENDIF
END SUB
!
SUB SEND.DATA
  CALL CMSEND(CONVERSATION.ID,      \
              BUFFER,                \
              SEND.LENGTH,           \
              REQUEST.TO.SEND.RECEIVED, \
              RETURN.CODE)
!
  PRINT #2; "Send_data rc = ",RETURN.CODE
  IF RETURN.CODE <> CM.OK THEN \
    BEGIN
      SYS.ERR = TRUE
      ERR.MSG = "Send.Data error."
      CALL DISP.MSG(ERR.MSG)
    ENDIF
END SUB
!
! Include CPI-Communications Pseudonym Equates
!
%INCLUDE CPIC_BAS.EQU
!
TRUE = 1
FALSE = 0
BUF.SIZE = 4096
SYS.ERR = FALSE
FILE.OPEN = FALSE
!
! Open log file for display message subroutine
!
ON ERROR GOTO ERROR.HANDLER
CREATE "ADX_UPGM:TPSCBAS.LOG" AS 2
!
CALL CMACCP(CONVERSATION.ID,      \
            RETURN.CODE)
!
IF RETURN.CODE <> CM.OK THEN \
  BEGIN
    SYS.ERR = TRUE
    ERR.MSG = "Accept.Conversation error."
    CALL DISP.MSG(ERR.MSG)
    STOP
  ENDIF
!
CALL RECEIVE
!
IF NOT SYS.ERR THEN \
  BEGIN
    IF DATA.RECEIVED = CM.COMPLETE.DATA.RECEIVED THEN \
      BEGIN
        FILE.NAME.LEN = RECEIVED.LENGTH
        FILE.NAME = MID$(BUFFER, 1, FILE.NAME.LEN)
        CALL DISP.MSG(FILE.NAME)
        CALL RECEIVE
        IF NOT SYS.ERR THEN \
```

```

BEGIN
IF STATUS.RECEIVED = CM.SEND.RECEIVED THEN \
  BEGIN
  FILE.OPEN = TRUE
  ! OPEN FILE.NAME DIRECT RECL BUF.SIZE AS 1 BUFFSIZE BUF.SIZE
  OPEN FILE.NAME RECL BUF.SIZE AS 1 BUFFSIZE BUF.SIZE
  ENDIF \
ELSE \
  BEGIN
  SYS.ERR = TRUE
  ERR.MSG = "Permission to send not received."
  CALL DISP.MSG(ERR.MSG)
  ENDIF
ENDIF
ENDIF \
ELSE \
  BEGIN
  SYS.ERR = TRUE
  ERR.MSG = "Complete request not received."
  CALL DISP.MSG(ERR.MSG)
  ENDIF
ENDIF
!
IF FILE.OPEN = TRUE THEN \
  BEGIN

  ! Calculate number of 4K blocks to transfer
  FILE.SIZE = SIZE(FILE.NAME)
  NUM.OF.4K.BLOCKS = INT%(FILE.SIZE/BUF.SIZE)

  ! Determine size of last block to send
  LAST.BLOCK = MOD(FILE.SIZE, BUF.SIZE)

  ! Read file and send 4K blocks
  SEND.LENGTH = BUF.SIZE
  FOR I% = 1 TO NUM.OF.4K.BLOCKS
    READ FORM "C4096"; #1, I%; BUFFER
    CALL DISP.MSG("Issuing Send_Data")
    CALL SEND.DATA
  NEXT I%

  ! Send last block or only block if file size < 4K
  ! GET data one byte at a time because record < 4K
  ! First clear send buffer
  BUFFER = ""
  IF LAST.BLOCK <> 0 THEN \
    BEGIN
    FOR I% = 1 TO LAST.BLOCK
      BYTE% = GET(1)
      BUFFER = BUFFER + CHR$(BYTE%)
    NEXT I%
    CALL DISP.MSG("Issuing Last Send_Data")
    CALL DISP.MSG(LAST.BLOCK$)
    SEND.LENGTH = LAST.BLOCK
    CALL SEND.DATA
    ENDIF
  !
  CLOSE 1
ENDIF \
ELSE \
  BEGIN
  SYS.ERR = TRUE
  !
  CALL DISP.MSG("Issuing Send_Error")
  CALL SEND.ERROR

```

Server TP for BASIC

```
!
ERR.MSG = "Unable to open requested file."
CALL DISP.MSG(ERR.MSG)
!
CALL DISP.MSG("Issuing Set_Deal_Type")
CALL CMSDT(CONVERSATION.ID,      \
            CM.DEALLOCATE.ABEND, \
            RETURN.CODE)
ENDIF
!
CALL DISP.MSG("Issuing Deallocate")
CALL CMDEAL(CONVERSATION.ID,      \
            RETURN.CODE)
!
CLOSE 2
STOP
!
ERROR.HANDLER:
CALL DISP.MSG("In Error Handler")
IF ERR = "OE" THEN \
    BEGIN
    FILE.OPEN = FALSE
    SYS.ERR = TRUE
    ERR.MSG = "Unable to open requested file."
    CALL DISP.MSG(ERR.MSG)
    RESUME
    ENDIF \
ELSE \
    BEGIN
    PRINT #2; "ERR = ",ERR
    READ.ERROR$ = MID$( ERR, 1, 1 )
    IF READ.ERROR$ = "R" THEN \
        BEGIN
        SYS.ERR = TRUE
        ERR.MSG = "Error reading requested file"
        CALL DISP.MSG(ERR.MSG)
        RESUME
        ENDIF \
    ENDIF
!
STOP
!
END
```

COBOL TPs

This section provides example information for transaction programs written in COBOL.

Example Header File Definitions for COBOL Transaction Programs

```
=====
01 CONVERSATION-ID          PIC X(8).
*
01 CONVERSATION-TYPE        PIC 9(9) COMP-5.
   88 CM-BASIC-CONVERSATION VALUE 0.
   88 CM-MAPPED-CONVERSATION VALUE 1.
*
01 CM-RETCODE               PIC 9(9) COMP-5.
*
   88 CM-OK                  VALUE 0.
   88 CM-ALLOCATE-FAILURE-NO-RETRY VALUE 1.
   88 CM-ALLOCATE-FAILURE-RETRY  VALUE 2.
   88 CM-CONVERSATION-TYPE-MISMATCH VALUE 3.
   88 CM-SECURITY-NOT-VALID      VALUE 6.
   88 CM-SYNC-LVL-NOT-SUPPORTED-PGM VALUE 8.
   88 CM-TPN-NOT-RECOGNIZED      VALUE 9.
   88 CM-TP-NOT-AVAILABLE-NO-RETRY VALUE 10.
```

```

88 CM-TP-NOT-AVAILABLE-RETRY      VALUE 11.
88 CM-DEALLOCATED-ABEND           VALUE 17.
88 CM-DEALLOCATED-NORMAL          VALUE 18.
88 CM-PARAMETER-ERROR             VALUE 19.
88 CM-PRODUCT-SPECIFIC-ERROR      VALUE 20.
88 CM-PROGRAM-ERROR-NO-TRUNC       VALUE 21.
88 CM-PROGRAM-ERROR-PURGING       VALUE 22.
88 CM-PROGRAM-ERROR-TRUNC         VALUE 23.
88 CM-PROGRAM-PARAMETER-CHECK     VALUE 24.
88 CM-PROGRAM-STATE-CHECK         VALUE 25.
88 CM-RESOURCE-FAILURE-NO-RETRY    VALUE 26.
88 CM-RESOURCE-FAILURE-RETRY      VALUE 27.
88 CM-UNSUCCESSFUL                VALUE 28.
*
01 DATA-RECEIVED                  PIC 9(9) COMP-5.
88 CM-NO-DATA-RECEIVED            VALUE 0.
88 CM-DATA-RECEIVED              VALUE 1.
88 CM-COMPLETE-DATA-RECEIVED      VALUE 2.
88 CM-INCOMPLETE-DATA-RECEIVED    VALUE 3.
*
01 DEALLOCATE-TYPE                 PIC 9(9) COMP-5.
88 CM-DEALLOCATE-SYNC-LEVEL       VALUE 0.
88 CM-DEALLOCATE-FLUSH           VALUE 1.
88 CM-DEALLOCATE-CONFIRM         VALUE 2.
88 CM-DEALLOCATE-ABEND           VALUE 3.
*
01 ERROR-DIRECTION                PIC 9(9) COMP-5.
88 CM-RECEIVE-ERROR              VALUE 0.
88 CM-SEND-ERROR                 VALUE 1.
*
01 CM-FILL                        PIC 9(9) COMP-5.
88 CM-FILL-LL                    VALUE 0.
88 CM-FILL-BUFFER                VALUE 1.
*
01 LOG-DATA                       PIC X(512).
*
01 LOG-DATA-LENGTH                PIC 9(9) COMP-5.
*
01 MODE-NAME                      PIC X(8).
*
01 MODE-NAME-LENGTH               PIC 9(9) COMP-5.
*
01 PARTNER-LU-NAME                PIC X(17).
*
01 PARTNER-LU-NAME-LENGTH         PIC 9(9) COMP-5.
*
01 PREPARE-TO-RECEIVE-TYPE        PIC 9(9) COMP-5.
88 CM-PREPARE-TO-RECEIVE-SYNC-LVL VALUE 0.
88 CM-PREP-TO-RECEIVE-FLUSH       VALUE 1.
88 CM-PREP-TO-RECEIVE-CONFIRM     VALUE 2.
*
01 RECEIVE-TYPE                   PIC 9(9) COMP-5.
88 CM-RECEIVE-AND-WAIT            VALUE 0.
88 CM-RECEIVE-IMMEDIATE          VALUE 1.
*
01 REQUEST-TO-SEND-RECEIVED       PIC 9(9) COMP-5.
88 CM-REQ-TO-SEND-NOT-RECEIVED    VALUE 0.
88 CM-REQ-TO-SEND-RECEIVED        VALUE 1.
*
01 RETURN-CONTROL                 PIC 9(9) COMP-5.
88 CM-WHEN-SESSION-ALLOCATED      VALUE 0.
88 CM-IMMEDIATE                   VALUE 1.
*
01 SEND-TYPE                      PIC 9(9) COMP-5.
88 CM-BUFFER-DATA                 VALUE 0.
88 CM-SEND-AND-FLUSH              VALUE 1.
88 CM-SEND-AND-CONFIRM            VALUE 2.

```

Header File Defs for COBOL TP

```
      88 CM-SEND-AND-PREP-TO-RECEIVE          VALUE 3.
      88 CM-SEND-AND-DEALLOCATE                VALUE 4.
*
01  STATUS-RECEIVED                          PIC 9(9) COMP-5.
      88 CM-NO-STATUS-RECEIVED                VALUE 0.
      88 CM-SEND-RECEIVED                    VALUE 1.
      88 CM-CONFIRMED-RECEIVED               VALUE 2.
      88 CM-CONFIRM-SEND-RECEIVED            VALUE 3.
      88 CM-CONFIRM-DEALLOC-RECEIVED          VALUE 4.
*
01  SYNC-LEVEL                              PIC 9(9) COMP-5.
      88 CM-NONE                             VALUE 0.
      88 CM-CONFIRM                          VALUE 1.
*
01  SYM-DEST-NAME                           PIC X(8).
*
01  TP-NAME                                 PIC X(64).
*
01  TP-NAME-LENGTH                         PIC 9(9) COMP-5  VALUE ZERO.
*
01  CM-REQUEST-TO-SEND-RECEIVED            PIC 9(9) COMP-5.
*
01  CPIC-RC                               PIC 9(9) COMP-5.
*
01  VERB-ID                               PIC 9(9) COMP-5.
*
01  ERRCODE-1                             PIC S9(11) COMP-5  VALUE 0.
*
01  ERRCODE-2                             PIC S9(11) COMP-5  VALUE 0.
=====
```

Example CPI for Communications File Requester Program

```
ID DIVISION.
PROGRAM-ID. TPRCOBOL.
*****
*
* Sample CPIC File Requester Program TPRCOBOL.CBL for 4690 OS
*
* This sample file requester transaction program allocates a
* conversation with file server TPSCOBOL to transfer a file
* specified at the prompt. It can be invoked from command mode
* by issuing TPRCOBOL. The prompt asks for the file to read.
* The specified file must reside on the server machine.
*
* Note : In order for this program to work correctly, the
* requested file must contain records of EXACTLY 80
* bytes in length. The file size will be a
* multiple of 80.
*
* The CPIC calls used in this transaction program are :
*
*          CMINIT (Initialize_Conversation)
*          CMSSL  (Set_Sync_Level)
*          CMALLC (Allocate_Conversation)
*          CMSEND (Send_Data)
*          CMCFM  (Confirm)
*          CMRCV  (Receive_Data)
*          CMCFMD (Confirmed)
*
* A Symbolic Destination Name of TPRCOBOL must be configured in
* 4690 Communications Configuration specifying Local LU,
* Partner LU, MODE, and Partner TP Name.
*
* This source code was compiled with the IBM COBOL/2 Compiler
* using the /litlink and /ans85 compiler options.
*
* The compiled source code was linked with the 4690 CPIC COBOL
```

```

* library ADXHSV1L.L86 shipped on the 4690 Optionals.      *
*                                                         *
*                                                         *
* (C) COPYRIGHT IBM CORP. 1990                             *
* LICENSED MATERIAL - PROGRAM PROPERTY OF IBM             *
* ALL RIGHTS RESERVED                                     *
*                                                         *
*****
ENVIRONMENT DIVISION.
SOURCE-COMPUTER. IBMPS2.
OBJECT-COMPUTER. IBMPS2.
FILE-CONTROL.
    SELECT FILE1 ASSIGN TO
    REQUESTED-FILE
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL.
*
DATA DIVISION.
FILE SECTION.
FD FILE1.
01 OUTPUT-LINE-WRITTEN.
   05 LINE-WRITTEN          PIC X(80).
*
WORKING-STORAGE SECTION.
* Copy standard COBOL declarations from CPIC_C.H
COPY "CPIC_CBL.H".
*
01 SEND-BUFFER              PIC X(80).
*
01 SEND-LENGTH              PIC 9(9) COMP-5   VALUE 80.
*
01 RECEIVE-BUFFER           PIC X(80).
*
01 RECEIVED-LENGTH          PIC 9(9) COMP-5   VALUE ZERO.
*
01 REQUESTED-LENGTH         PIC 9(9) COMP-5   VALUE ZERO.
*
01 REQUESTED-FILE.
   05 FILE-NAME              PIC X(64).

PROCEDURE DIVISION.
START-PARA.
    DISPLAY "Enter filename to read:".
    ACCEPT FILE-NAME.

    MOVE "TPRCOBOL" TO SYM-DEST-NAME.
    DISPLAY "ISSUING CMINIT".
    CALL "CMINIT" USING CONVERSATION-ID
                        SYM-DEST-NAME
                        CM-RETCODE.

* Set Sync Level = Confirm

    MOVE 1 TO SYNC-LEVEL.
    DISPLAY "ISSUING CMSSL".
    CALL "CMSSL" USING CONVERSATION-ID
                SYNC-LEVEL
                CM-RETCODE.

    DISPLAY "ISSUING CMALLC".
    CALL "CMALLC" USING CONVERSATION-ID
                CM-RETCODE.

* Send filename to server program

```

CPI-C Requester TP for COBOL

```
    DISPLAY "ISSUING CMSEND TO SEND FILENAME TO SERVER".
    MOVE FILE-NAME TO SEND-BUFFER.
    MOVE 64 TO SEND-LENGTH.
    CALL "CMSEND" USING CONVERSATION-ID
        SEND-BUFFER
        SEND-LENGTH
        REQUEST-TO-SEND-RECEIVED
        CM-RETCODE.

* Confirm the filename

    DISPLAY "ISSUING CMCFM".
    CALL "CMCFM" USING CONVERSATION-ID
        REQUEST-TO-SEND-RECEIVED
        CM-RETCODE.

    OPEN OUTPUT FILE1.
    MOVE 80 TO REQUESTED-LENGTH.

* Receive data until data-received is complete

    PERFORM RECEIVE-DATA UNTIL CM-CONFIRM-DEALLOC-RECEIVED.

* Issue Confirmation

    CALL "CMCFMD" USING CONVERSATION-ID
        CM-RETCODE.

* When transfer is complete, close file and end

    CLOSE FILE1.
    STOP RUN.

* * * * *
* Begin Receive Data routine *
* * * * *

RECEIVE-DATA.

    DISPLAY "ISSUING CMRCV".
    CALL "CMRCV" USING CONVERSATION-ID
        RECEIVE-BUFFER
        REQUESTED-LENGTH
        DATA-RECEIVED
        RECEIVED-LENGTH
        STATUS-RECEIVED
        REQUEST-TO-SEND-RECEIVED
        CM-RETCODE.

    IF NOT CM-OK PERFORM CLEAN-UP.

    IF CM-COMPLETE-DATA-RECEIVED
        MOVE RECEIVE-BUFFER TO LINE-WRITTEN
        WRITE OUTPUT-LINE-WRITTEN.

* * * * *
* End Receive Data routine *
* * * * *
```



```

CLEAN-UP.
  DISPLAY CM-RETCODE.
  DISPLAY "ERROR FOUND".
  STOP RUN.
=====

```

Example CPI for Communications File Server Program

```

ID DIVISION.
PROGRAM-ID. TPSCOBOL.
*****
*
* Sample CPIC File Server Program TPSCOBOL.CBL for 4690 OS
*
* This sample file server transaction program accepts a
* conversation allocated by partner transaction program
* TPRCOBOL. It is started as a 4690 Background Application.
* The requester first sends the file name of the requested file
* and this server program sends it to the requester. The file
* must exist or this program generates a run-time error.
*
* Note : In order for this program to work correctly, the
*         requested file must contain records of EXACTLY 80
*         bytes in length. The file size will be a
*         multiple of 80.
*
* The following CPIC calls are contained in this program :
*
*         CMACCP (Accept Conversation)
*         CMRCV  (Receive_Data)
*         CMCFMD (Confirmed)
*         CMSEND (Send_Data)
*         CMDEAL (DealToocate)
*
* A Remotely Attachable Local TP record must be configured in
* 4690 Controller Communications Configuration specifying a
* TP Name identical to that configured in the TPRCOBOL
* Symbolic Destination Name - Partner TP Name. The
* conversation type is Mapped and the sync level is Confirm.
* The filespec should contain the drive, subdirectory and
* filename of the executable copy of this server transaction
* program, e.g. C:\ADX_UPGM\TPSCOBOL.286.
* Also, the Remotely Attachable Local TP record name must be
* referenced in a Symbolic Destination Name record on the
* server controller.
*
* Note: Because background applications cannot write directly to
*       screen, all messages that this program generates are
*       written to a log file named TPSCOBOL.LOG created in
*       the root subdirectory.
*
* This source code was compiled with the IBM COBOL/2 Compiler
* using the /litlink and /ans85 compiler options.
*
* The compiled source code was linked with the 4690 CPIC COBOL
* library ADXHSV1L.L86 shipped on the 4690 Optionals.
*
*
* (C) COPYRIGHT IBM CORP. 1990
* LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
* ALL RIGHTS RESERVED
*
*****
ENVIRONMENT DIVISION.
SOURCE-COMPUTER. IBMP52.
OBJECT-COMPUTER. IBMP52.

```

CPI-C Server TP for COBOL

```
FILE-CONTROL.
  SELECT FILE1 ASSIGN TO
    DATA-NAME-1
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL.
*
  SELECT FILE2 ASSIGN TO
    LOG-FILE
    ORGANIZATION LINE SEQUENTIAL
    ACCESS SEQUENTIAL.
*
DATA DIVISION.
FILE SECTION.
FD FILE1.
01 INPUT-LINE-READ.
   05 LINE-READ          PIC X(80).
*
FD FILE2.
01 LOG-MESSAGE.
   05 MESSAGE-LOGGED     PIC X(80).
*
WORKING-STORAGE SECTION.
* Copy standard COBOL declarations from CPIC_CBL.H
COPY "CPIC_CBL.H".
*
01 SEND-BUFFER          PIC X(80).
*
01 SEND-LENGTH          PIC 9(9) COMP-5   VALUE 80.
*
01 RECEIVE-BUFFER       PIC X(512).
*
01 RECEIVED-LENGTH      PIC 9(9) COMP-5   VALUE ZERO.
*
01 REQUESTED-LENGTH     PIC 9(9) COMP-5   VALUE ZERO.
*
01 EOF                  PIC 9              VALUE ZERO.
*
01 LOG-FILE             PIC X(12)         VALUE "TPSCOBOL.LOG".
01 DATA-NAME-1.
   05 FILE-NAME         PIC X(100).
*
PROCEDURE DIVISION.
START-PARA.
  OPEN OUTPUT FILE2.
  MOVE ZEROS TO CM-RETCODE.

* Accept Conversation

  CALL "CMACCP" USING CONVERSATION-ID
    CM-RETCODE.

  IF (NOT CM-OK)
    MOVE "Error - CMACCP failed" TO MESSAGE-LOGGED
    WRITE LOG-MESSAGE
    PERFORM CLEAN-UP.

* Get filename from requester

  PERFORM RECEIVE-DATA UNTIL CM-COMPLETE-DATA-RECEIVED.
  MOVE RECEIVE-BUFFER TO FILE-NAME.

* Issue Receive until send_received

  PERFORM RECEIVE-DATA UNTIL CM-SEND-RECEIVED.

* When send_received then go into send loop
```

```

OPEN INPUT FILE1.
READ FILE1 AT END MOVE 1 TO EOF.
PERFORM SEND-DATA UNTIL EOF IS EQUAL TO 1.

* After sending all data, issue deallocate

CALL "CMDEAL" USING CONVERSATION-ID
                      CM-RETCODE.

CLOSE FILE1.
PERFORM CLEAN-UP.

* * * * *
* Begin Send Data routine *
* * * * *

SEND-DATA.
  MOVE LINE-READ TO SEND-BUFFER
  CALL "CMSEND" USING CONVERSATION-ID
                      SEND-BUFFER
                      SEND-LENGTH
                      REQUEST-TO-SEND-RECEIVED
                      CM-RETCODE.

  IF (NOT CM-OK)
    MOVE "Error - CMSEND failed" TO MESSAGE-LOGGED
    WRITE LOG-MESSAGE
    PERFORM CLEAN-UP.

  READ FILE1 AT END MOVE 1 TO EOF.

* * * * *
* End Send Data routine *
* * * * *

* * * * *
* Begin Receive Data routine *
* * * * *

RECEIVE-DATA.
  MOVE 512 TO REQUESTED-LENGTH.
  CALL "CMRCV" USING CONVERSATION-ID
                      RECEIVE-BUFFER
                      REQUESTED-LENGTH
                      DATA-RECEIVED
                      RECEIVED-LENGTH
                      STATUS-RECEIVED
                      REQUEST-TO-SEND-RECEIVED
                      CM-RETCODE.

  IF (NOT CM-OK)
    MOVE "Error - CMRCV failed" TO MESSAGE-LOGGED
    WRITE LOG-MESSAGE
    PERFORM CLEAN-UP.
  IF CM-CONFIRMED-RECEIVED
    CALL "CMCFMD" USING CONVERSATION-ID
                      CM-RETCODE.

* * * * *
* End Receive Data routine *
* * * * *

* * * * *
* Clean Up routine *
* * * * *
```

CPI-C Server TP for COBOL

```
CLEAN-UP.  
  CLOSE FILE2.  
  STOP RUN.
```

Appendix F. Notices

This information was developed for products and services offered in the U.S.A.

Toshiba Global Commerce Solutions may not offer the products, services, or features discussed in this document in other countries. Consult your local Toshiba Global Commerce Solutions representative for information on the products and services currently available in your area. Any reference to a Toshiba Global Commerce Solutions product, program, or service is not intended to state or imply that only that Toshiba Global Commerce Solutions product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Toshiba Global Commerce Solutions intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-Toshiba Global Commerce Solutions product, program, or service.

Toshiba Global Commerce Solutions may have patents or pending patent applications covering the subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Toshiba Global Commerce Solutions
Attn: General Counsel
3039 E. Cornwallis Rd
RTP, NC 27709

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: TOSHIBA GLOBAL COMMERCE SOLUTIONS PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Toshiba Global Commerce Solutions may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Toshiba Global Commerce Solutions may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this information to non-Toshiba Global Commerce Solutions Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Toshiba Global Commerce Solutions product and use of those Web sites is at your own risk.

Information concerning non-Toshiba Global Commerce Solutions products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Toshiba Global Commerce Solutions has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Toshiba Global Commerce Solutions products. Questions on the capabilities of non-Toshiba Global Commerce Solutions products should be addressed to the suppliers of those products.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Telecommunication regulatory statement

This product is not intended to be connected directly or indirectly by any means whatsoever to interfaces of public telecommunications networks, nor is it intended to be used in a public services network.

Electronic Emission Notices

When you attach a monitor to the equipment, you must use the designated monitor cable and any interference suppression devices that are supplied with the monitor.

Federal Communications Commission (FCC) Statement

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user will be required to correct the interference at his own expense.

Properly shielded and grounded cables and connectors must be used in order to meet FCC emission limits. Toshiba Global Commerce Solutions is not responsible for any radio or television interference caused by using other than recommended cables and connectors or by unauthorized changes or modifications to this equipment. Unauthorized changes or modifications could void the user's authority to operate the equipment.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference, and
2. This device must accept any interference received, including interference that may cause undesired operation.

Industry Canada Class A Emission Compliance statement

This Class A digital apparatus complies with Canadian ICES-003.

Avis de conformité à la réglementation d'Industrie Canada

Cet appareil numérique de la classe A est conforme à la norme NMB-003 du Canada.

Australia and New Zealand Class A Statement

Attention: This is a Class A product. In a domestic environment this product may cause radio interference, in which case the user may be required to take adequate measures.

European Union Electromagnetic Compatibility (EMC) Directive Conformance Statement

This product is in conformity with the protection requirements of EU Council Directive 2004/108/EC on the approximation of the laws of the Member States relating to electromagnetic compatibility. Toshiba Global Commerce Solutions cannot accept responsibility for any failure to satisfy the protection requirements resulting from a non-recommended modification of the product, including the fitting of non-Toshiba Global Commerce Solutions option cards.

This product has been tested and found to comply with the limits for Class A Information Technology Equipment according to CISPR 22/European Standard EN 55022. The limits for Class A equipment were derived for commercial and industrial environments to provide reasonable protection against interference with licensed communication equipment.

Attention: This is a Class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

Responsible manufacturer:

Toshiba Global Commerce Solutions
3039 Cornwallis Road
Building 307
Research Triangle Park, North Carolina 27709
United States of America

European Community contact:

Toshiba Global Commerce Solutions
Brand Manager - Europe, Middle East & Africa
3 NEW SQUARE, FELTHAM, TW14 8HB Great Britain
Building: 1 Floor: NA | Office: MOBILE
Tel: 44-7967-275819
e-mail: robin_lyon@uk.ibm.com

Germany Class A Statement

Deutschsprachiger EU Hinweis: Hinweis für Geräte der Klasse A EU-Richtlinie zur Elektromagnetischen Verträglichkeit

Dieses Produkt entspricht den Schutzanforderungen der EU-Richtlinie 2004/108/EG zur Angleichung der Rechtsvorschriften über die elektromagnetische Verträglichkeit in den EU-Mitgliedsstaaten und hält die Grenzwerte der EN 55022 Klasse A ein.

Um dieses sicherzustellen, sind die Geräte wie in den Handbüchern beschrieben zu installieren und zu betreiben. Des Weiteren dürfen auch nur von der Toshiba Global Commerce Solutions empfohlene Kabel angeschlossen werden. Toshiba Global Commerce Solutions übernimmt keine Verantwortung für die Einhaltung der Schutzanforderungen, wenn das Produkt ohne Zustimmung der Toshiba Global Commerce Solutions verändert bzw. wenn Erweiterungskomponenten von Fremdherstellern ohne Empfehlung der Toshiba Global Commerce Solutions gesteckt/eingebaut werden.

EN 55022 Klasse A Geräte müssen mit folgendem Warnhinweis versehen werden: "Warnung: Dieses ist eine Einrichtung der Klasse A. Diese Einrichtung kann im Wohnbereich Funk-Störungen verursachen; in diesem Fall kann vom Betreiber verlangt werden, angemessene Maßnahmen zu ergreifen und dafür aufzukommen."

Deutschland: Einhaltung des Gesetzes über die elektromagnetische Verträglichkeit von Geräten

Dieses Produkt entspricht dem "Gesetz über die elektromagnetische Verträglichkeit von Geräten (EMVG)". Dies ist die Umsetzung der EU-Richtlinie 2004/108/EG in der Bundesrepublik Deutschland.

Zulassungsbescheinigung laut dem Deutschen Gesetz über die elektromagnetische Verträglichkeit von Geräten (EMVG) (bzw. der EMC EG Richtlinie 2004/108/EG) für Geräte der Klasse A

Dieses Gerät ist berechtigt, in Übereinstimmung mit dem Deutschen EMVG das EG-Konformitätszeichen - CE - zu führen.

Verantwortlich für die Einhaltung der EMV Vorschriften ist der Hersteller:

Toshiba Global Commerce Solutions
3039 Cornwallis Road
Building 307
Research Triangle Park, North Carolina 27709
United States of America

Der verantwortliche Ansprechpartner des Herstellers in der EU ist:

Toshiba Global Commerce Solutions
Brand Manager - Europe, Middle East & Africa
3 NEW SQUARE, FELTHAM, TW14 8HB Great Britain
Building: 1 Floor: NA | Office: MOBILE
Tel: 44-7967-275819
e-mail: robin_lyon@uk.ibm.com

Generelle Informationen:

Das Gerät erfüllt die Schutzanforderungen nach EN 55024 und EN 55022 Klasse A.

Japan Voluntary Control Council for Interference Class A statement

Attention: This is a Class A product based on the standard of the Voluntary Control Council for Interference (VCCI). If this equipment is used in a domestic environment, radio interference may occur, in which case, the user may be required to take corrective actions.

この装置は、クラスA情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。 VCCI-A

Japan Electronics and Information Technology Industries Association (JEITA) statement

高調波ガイドライン準用品

Japan Electronics and Information Technology Industries Association (JEITA) Confirmed Harmonics Guidelines with Modifications (products greater than 20 A per phase)

Korean communications statement

Please note that this device has been approved for business purposes with regard to electromagnetic interference (Type A). If you find this is not suitable for your use, you may exchange it for a non-business purpose one.

이 기기는 업무용(A급)으로 전자파적합기기로서 판매자 또는 사용자는 이 점을 주의하시기 바라며, 가정외의 지역에서 사용하는 것을 목적으로 합니다.

Russian Electromagnetic Interference (EMI) Class A statement

ВНИМАНИЕ! Настоящее изделие относится к классу А. В жилых помещениях оно может создавать радиопомехи, снижения которых необходимы дополнительные меры

People's Republic of China Class A electronic emission Statement

Attention: This is a Class A product. In a domestic environment this product may cause radio interference, in which case the user may be required to take adequate measures.

中华人民共和国“A类”警告声明

声 明

此为 A 级产品，在生活环境中，该产品可能会造成无线电干扰。在这种情况下，可能需要用户对其干扰采取切实可行的措施。

Taiwan Class A compliance statement

警告使用者：

這是甲類的資訊產品，在居住的環境中使用時，可能會造成射頻干擾，在這種情況下，使用者會被要求採取某些適當的對策。

European Community (EC) Mark of Conformity Statement

This product is in conformity with the protection requirements of EC Council Directive 89/336/EEC on the approximation of the laws of the Member States relating to electromagnetic compatibility. Toshiba Global Commerce Solutions cannot accept responsibility for any failure to satisfy the protection requirements resulting from a non-recommended modification of the product, including the fitting of non-Toshiba Global Commerce Solutions option cards.

This product has been tested and found to comply with the limits for Class A Information Technology Equipment according to CISPR 22 / European Standard EN 55022. The limits for Class A equipment were derived for commercial and industrial environments to provide reasonable protection against interference with licensed communication equipment.

Warning: This is a Class A product. In a domestic environment this product may cause radio interference, in which case the user may be required to take adequate measures.

Electrostatic Discharge (ESD)

Attention: Electrostatic discharge (ESD) damage can occur when there is a difference in charge between the part, the product, and the service person. No damage will occur if the service person and the part being installed are at the same charge level.

ESD damage prevention

Anytime a service action involves physical contact with logic cards, modules, back-panel pins, or other ESD sensitive (ESDS) parts, the service person must be connected to an ESD common ground point on the product through the ESD wrist strap and cord.

The ESD ground clip can be attached to any frame ground, ground braid, green wire ground, or the round ground prong on the AC power plug. Coax or connector outside shells can also be used.

Handling removed cards

Logic cards removed from a product should be placed in ESD protective containers. No other object should be allowed inside the ESD container with the logic card. Attach tags or reports that must accompany the card to the outside of the container.

Japanese Electrical Appliance and Material Safety Law statement

本製品およびオプションに電源コードセットが付属する場合は、それぞれその装置専用のものになっていますので他の機器には使用しないで下さい。

Japanese power line harmonics compliance statement

高調波ガイドライン適合品

高調波ガイドライン適合品

Cable ferrite requirement

All cable ferrites are required to suppress radiated EMI emissions and must not be removed.

Product recycling and disposal

This unit must be recycled or discarded according to applicable local and national regulations. Toshiba Global Commerce Solutions encourages owners of information technology (IT) equipment to responsibly recycle their equipment when it is no longer needed. Toshiba Global Commerce Solutions offers a variety of product return programs and services in several countries to assist equipment owners in recycling their IT products. Information on Toshiba Global Commerce Solutions product recycling offerings can be found on the Toshiba Global Commerce Solutions product recycling web site.

Español:

Esta unidad debe reciclarse o desecharse de acuerdo con lo establecido en la normativa nacional o local aplicable. Toshiba Global Commerce Solutions recomienda a los propietarios de equipos de tecnología de la información (TI) que reciclen responsablemente sus equipos cuando éstos ya no les sean útiles. Toshiba Global Commerce Solutions dispone de una serie de programas y servicios de devolución de productos en varios países, a fin de ayudar a los propietarios de equipos a reciclar sus productos de TI. Se puede encontrar información sobre las ofertas de reciclado de productos de Toshiba Global Commerce Solutions en el sitio web Toshiba Global Commerce Solutions product recycling.



Notice: This mark applies only to countries within the European Union (EU) and Norway.

Appliances are labeled in accordance with European Directive 2002/96/EC concerning waste electrical and electronic equipment (WEEE). The Directive determines the framework for the return and recycling of used appliances as applicable throughout the European Union. This label is applied to various products to indicate that the product is not to be thrown away, but rather reclaimed upon end of life per this Directive.

Remarque : Cette marque s'applique uniquement aux pays de l'Union Européenne et à la Norvège. L'étiquette du système respecte la Directive européenne 2002/96/EC en matière de Déchets des Equipements Electriques et Electroniques (DEEE), qui détermine les dispositions de retour et de recyclage applicables aux systèmes utilisés à travers l'Union européenne. Conformément à la directive, ladite étiquette précise que le produit sur lequel elle est apposée ne doit pas être jeté mais être récupéré en fin de vie.

注意：このマークは EU 諸国およびノルウェーにおいてのみ適用されます。

この機器には、EU 諸国に対する廃電気電子機器指令 2002/96/EC(WEEE) のラベルが貼られています。この指令は、EU 諸国に適用する使用済み機器の回収とリサイクルの骨子を定めています。このラベルは、使用済みになった時に指令に従って適正な処理をする必要があることを知らせるために種々の製品に貼られています。

In accordance with the European WEEE Directive, electrical and electronic equipment (EEE) is to be collected separately and to be reused, recycled, or recovered at end of life. Users of EEE with the WEEE marking per Annex IV of the WEEE Directive, as shown above, must not dispose of end of life EEE as unsorted municipal waste, but use the collection framework available to customers for the return, recycling, and recovery of WEEE. Customer participation is important to minimize any potential effects of EEE on the environment and human health due to the potential presence of hazardous substances in EEE. For proper collection and treatment, contact your local Toshiba Global Commerce Solutions representative.

Disposal of IT products should be in accordance with local ordinances and regulations.

Battery return program

This product may contain sealed lead acid, nickel cadmium, nickel metal hydride, lithium, or lithium ion battery. Consult your user manual or service manual for specific battery information. The battery must be recycled or disposed of properly. Recycling facilities may not be available in your area. For information on disposal of batteries go to the Battery disposal web site or contact your local waste disposal facility.

For Taiwan:



Please recycle batteries.

For the European Union:



Notice: This mark applies only to countries within the European Union (EU)

Batteries or packaging for batteries are labeled in accordance with European Directive 2006/66/EC concerning batteries and accumulators and waste batteries and accumulators. The Directive determines the framework for the return and recycling of used batteries and accumulators as applicable throughout the European Union. This label is applied to various batteries to indicate that the battery is not to be thrown away, but rather reclaimed upon end of life per this Directive.

Les batteries ou emballages pour batteries sont étiquetés conformément aux directives européennes 2006/66/EC, norme relative aux batteries et accumulateurs en usage et aux batteries et accumulateurs usés. Les directives déterminent la marche à suivre en vigueur dans l'Union Européenne pour le retour et le recyclage des batteries et accumulateurs usés. Cette étiquette est appliquée sur diverses batteries pour indiquer que la batterie ne doit pas être mise au rebut mais plutôt récupérée en fin de cycle de vie selon cette norme.

バッテリーあるいはバッテリー用のパッケージには、EU 諸国に対する廃電気電子機器指令 2006/66/EC のラベルが貼られています。この指令は、バッテリーと蓄電池、および廃棄バッテリーと蓄電池に関するものです。この指令は、使用済みバッテリーと蓄電池の回収とリサイクルの骨子を定めているもので、EU 諸国にわたって適用されます。このラベルは、使用済みになったときに指令に従って適正な処理をする必要があることを知らせるために種々のバッテリーに貼られています。

In accordance with the European Directive 2006/66/EC, batteries and accumulators are labeled to indicate that they are to be collected separately and recycled at end of life. The label on the battery may also include a chemical symbol for the metal concerned in the battery (Pb for lead, Hg for mercury and Cd for

cadmium). Users of batteries and accumulators must not dispose of batteries and accumulators as unsorted municipal waste, but use the collection framework available to customers for the return, recycling and treatment of batteries and accumulators. Customer participation is important to minimize any potential effects of batteries and accumulators on the environment and human health due to the potential presence of hazardous substances. For proper collection and treatment, contact your local Toshiba Global Commerce Solutions representative.

This notice is provided in accordance with Royal Decree 106/2008 of Spain: The retail price of batteries, accumulators and power cells includes the cost of the environmental management of their waste.

For California:

Perchlorate material – special handling may apply

Refer to California Department of Toxic Substances Control.

The foregoing notice is provided in accordance with *California Code of Regulations Title 22, Division 4.5, Chapter 33: Best Management Practices for Perchlorate Materials*. This product/part includes a lithium manganese dioxide battery which contains a perchlorate substance.

Flat panel displays

The fluorescent lamp in the liquid crystal display contains mercury. Dispose of it as required by local ordinances and regulations.

Monitors and workstations

Connecticut: Visit the website of the Department of Energy & Environmental Protection for information about recycling covered electronic devices in the State of Connecticut, or telephone the Connecticut Department of Environmental Protection at 1-860-424-3000.

Oregon: For information regarding recycling covered electronic devices in the state of Oregon, go to the Oregon Department of Environmental Quality site.

Washington: For information about recycling covered electronic devices in the State of Washington, go to the Department of Ecology Web site or telephone the Washington Department of Ecology at 1-800-Recycle.

Trademarks

The following are trademarks or registered trademarks of Toshiba, Inc. in the United States or other countries, or both:

Toshiba
The Toshiba logo

The following are trademarks of Toshiba Global Commerce Solutions in the United States or other countries, or both:

AnyPlace
SureMark
SurePoint
SurePOS
TCxWave

The following are trademarks of International Business Machines Corporation in the United States or other countries, or both:

DB2
DB2 Universal Database
IBM and the IBM logo
PS/2
Wake on LAN
WebSphere

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Magellan is a registered trademark of Datalogic Scanning, Inc.

SYMBOL a registered trademark of Symbol Technologies, Inc.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Celeron and Intel are trademarks of Intel corporation in the United States, or other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, or other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

This glossary includes terms and definitions from the *IBM Dictionary of Computing* (New York; McGraw-Hill, Inc., 1994).

A

ABM. Asynchronous balanced mode.

access method. A software component in a processor for controlling the flow of information through a network.

ACF/VTAM. Advanced Communications Function for the Virtual Telecommunications Access Method.

active. (1) Able to communicate on the network. A token-ring network adapter is active if it is able to transmit and receive on the network. (2) Operational. (3) Pertaining to a node or device that is connected or is available for connection to another node or device. (4) Currently transmitting or receiving.

adapter. (1) In the point-of-sale terminal, a circuit card that, with its associated software, enables the terminal to use a function or feature. (2) In a LAN, within a communicating device, a circuit card that, with its associated software and/or microcode, enables the device to communicate over the network.

adapter address. Twelve hexadecimal digits that identify a LAN adapter.

ADCS. Advanced Data Communications for Stores

address. (1) In data communication, the IEEE-assigned unique code or the unique locally administered code assigned to each device or workstation connected to a network. (2) A character, group of characters, or a value that identifies a register, a particular part of storage, a data source, or a data link. The value is represented by one or more characters. (3) To refer to a device or an item of data by its address. (4) The location in the storage of a computer where data is stored.

Advanced Data Communications for Stores (ADCS). An IBM-licensed product that functions at the host processor to permit host-to-store communication.

alert. (1) An error message sent to the system services control point (SSCP) at the host system. (2) For LAN management products, a notification indicating a possible security violation, a persistent error condition, or an interruption or potential interruption in the flow of data around the network. See also *network management vector transport*. (3) In SNA, a record sent to a system problem management focal point to communicate the existence of an alert condition. (4) In the NetView program, a high-priority event that warrants

immediate attention. This data base record is generated for certain event types that are designed by user-constructed filters.

alphanumeric. Pertaining to a character set containing letters, digits, and other special characters.

Alphanumeric point-of-sale keyboard (ANPOS keyboard). This keyboard consists of a section of alphanumeric keys, a programmable set of point-of-sale keys, a numeric keypad, and system function keys. If attached through the PS/2 port, this keyboard can optionally include a pointing device.

alternate adapter. In a personal computer that is used on a LAN and that supports installation of two network adapters, the adapter that uses alternate (not standard or default) mapping between adapter-shared RAM, adapter ROM, and designated computer memory segments. The alternate adapter is usually designated as adapter 1 in configuration parameters. Contrast with *primary adapter*.

Alternate File Server. A store controller that maintains image versions of all non-system mirrored files and that can assume control if the configured File Server becomes disabled.

Alternate Master Store Controller. The store controller that can take control of the LAN if the configured Master Store Controller becomes disabled. It maintains image versions of both system mirrored and system compound files.

American National Standard Code for Information Interchange (ASCII). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphics characters.

ANPOS keyboard. Alphanumeric point-of-sale keyboard.

API. Application program interface.

application program. (1) A program written for or by a user that applies to the user's own work. (2) A program written for or by a user that applies to a particular application. (3) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

application program interface (API). The formally defined programming language interface that is between a Toshiba system control program or a licensed program and the user of the program.

architecture. A logical structure that encompasses operating principles including services, functions, and protocols. See *computer architecture*, *network architecture*, *Systems Application Architecture (SAA)*, *Systems Network Architecture (SNA)*.

ARTIC adapter. A family of communications coprocessor adapters that, with appropriate electrical interfaces, can support a wide range of communication devices. For the Tohsiba Store System, an ARTIC adapter provides communications support for ASYNC, SDLC, and X.25 communications.

ASCII. American National Standard Code for Information Interchange.

async. asynchronous.

asynchronous (async). (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as timing signals. (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions.

asynchronous balanced mode (ABM). An operational mode of a balanced data link in which either combined station can send commands at any time and can initiate transmission of response frames without explicit permission from the other combined station.

attach. (1) To connect a device physically. (2) To make a device a part of a network logically. Compare with *connect*.

attaching device. Any device that is physically connected to a network and can communicate over the network.

B

background. On a color display, the part of the display screen that surrounds a character.

background application. A non-interactive program that can be selected from the background application screen or that can start automatically when the system is IPLed or when the controller is activated as the master or file server. Contrast with *foreground application*.

backup. Pertaining to a system, device, file, or facility that can be used in the event of a malfunction or the loss of data.

bar code. A code representing characters by sets of parallel bars of varying thickness and separation that are read optically by transverse scanning.

baseband. (1) A frequency band that uses the complete bandwidth of a transmission medium. Contrast with *broadband*, *carrierband*. (2) A method of data transmission that encodes, modulates, and impresses

information on the transmission medium without shifting or altering the frequency of the information signal.

base unit. The part of the 4683 Point-of-Sale terminal that contains the power supply and the interfaces.

BASIC. Beginner's All-purpose Symbolic Instruction Code. A programming language that uses common English words.

basic conversation. A conversation in which programs exchange data records in an SNA-defined format. This format is a stream of data containing 2-byte length prefixes that specify the amount of data to follow before the next prefix.

batch. Smaller subdivisions of price change records within an event. Each batch has a 12-character ID and a 30-character description field.

baud. The rate at which signal conditions are transmitted per second. Contrast with *bits per second (bps)*.

beacon. (1) A frame sent by an adapter on a ring network indicating a serious ring problem, such as a broken cable. It contains the addresses of the beaconing station and its nearest active upstream neighbor (NAUN). (2) To send beacon frames continuously. An adapter is *beaconing* if it is sending such a frame.

beaconing. An error-indicating function of token-ring adapters that assists in locating a problem causing a hard error on a token-ring network.

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Pertaining to a selection, choice, or condition that has two possible different values or states.

bind. In SNA products, a request to activate a session between two logical units.

BIND. See bind session.

bind session (BIND). In SNA products, a request to activate a session between two logical units (LUs).

bit. Either of the binary digits: a 0 or 1.

bits per second (bps). The rate at which bits are transmitted per second. Contrast with *baud*.

block size. (1) The minimum size that frames are grouped into for retransmission. (2) The number of data elements (such as bits, bytes, characters, or records) that are recorded or transmitted as a unit.

bootstrap. A sequence of instructions whose execution causes additional instructions to be loaded and executed until the complete computer program is in storage.

bps. Bits per second.

Bps. Bytes per second.

bridge. (1) An attaching device connected to two LAN segments to allow the transfer of information from one LAN segment to the other. A bridge may connect the LAN segments directly by network adapters and software in a single device, or may connect network adapters in two separate devices through software and use of a telecommunications link between the two adapters. (2) A functional unit that connects two LANs that use the same logical link control (LLC) procedures but may use the same or different medium access control (MAC) procedures. Contrast with *gateway* and *router*.

broadband. A frequency band divisible into several narrower bands so that different kinds of transmissions such as voice, video, and data transmission can occur at the same time. Synonymous with *wideband*. Contrast with *baseband*.

buffer. (1) A portion of storage used to hold input or output data temporarily. (2) A routine or storage used to compensate for a difference in data rate or time of occurrence of events, when transferring data from one device to another.

bus. (1) In a processor, a physical facility on which data is transferred to all destinations, but from which only addressed destinations may read in accordance with appropriate conventions. (2) A network configuration in which nodes are interconnected through a bidirectional transmission medium. (3) One or more conductors used for transmitting signals or power.

byte. A string consisting of 8 bits that is treated as a unit, and that represents a character. See *n-bit byte*.

C

C. A high-level programming language designed to optimize run time, size, and efficiency.

C & SM. Communications and systems management.

cable loss (optical). The loss in an optical cable equals the attenuation coefficient for the cables fiber times the cable length.

cable segment. A section of cable between components or devices on a network. A segment may consist of a single patch cable, multiple patch cables connected together, or a combination of building cable and patch cables connected together. See *LAN segment*, *ring segment*.

call. The action of bringing a function or subprogram into effect, usually by specifying the entry conditions and jumping to an entry point.

carrierband. A frequency band in which the modulated signal is superimposed on a carrier signal (as differentiated from baseband), but only one channel is present on the medium. Contrast with *baseband*, *broadband*.

cash drawer. A drawer at a point-of-sale terminal that can be programmed to open automatically. See *till*.

CCB. Command control block.

CCC/IP. Controller-to-Controller Communications over Internet Protocol.

CCITT. Comité Consultatif International Télégraphique et Téléphonique. The International Telegraph and Telephone Consultative Committee.

CD. Corrective diskette.

CD-ROM. Compact disc Read-only memory. High-capacity read-only memory in the form of an optically read compact disk.

chain. (1) Transfer of control from the currently executing program to another program or overlay. (2) Referencing a data record from a previous data record.

channel. (1) A functional unit, controlled by a host computer, that handles the transfer of data between processor storage and local peripheral equipment. (2) A path along which signals can be sent. (3) The portion of a storage medium that is accessible to a given reading or writing station.

CICS. Customer Information Control System.

circuit. (1) A logic device. (2) One or more conductors through which an electric current can flow.

class. (1) A template for creating objects; a class defines data and methods; a class is a unit of organization in a Java program. A class can pass on its public data and methods to its subclasses. (2) A collection of variables and methods that an object can have, or a template for building objects.

.class file. A file containing machine-independent Java bytecodes. The Java compiler generates *.class* files for the Java interpreter to read.

class method. A class method is a function that is defined as a part of a class.

classpath. An environment variable used to define all the directories where *.class* files are found.

.class variable. A variable allocated once per class. Class variables have global class scope and belong to the entire class instead of an instance.

clear. To delete data from a screen or from memory.

COBOL. Common business-oriented language. A high-level programming language, based on English, that is used primarily for business applications.

command. (1) A request for performance of an operation or execution of a program. (2) A character string from a source external to a system that represents a request for system action.

command control block (CCB). In the Token-Ring Network, a specifically formatted information provided from the application program to the adapter support software to request an operation.

Common Programming Interface-Communications (CPI-C). Provides languages, commands, and calls that allow the development of applications that are more easily integrated and moved across environments supported by Systems Applications Architecture (SAA).

communication adapter. A circuit card and its associated software that enable a device, such as a personal computer, to be connected to a network or another computer (examples include binary synchronous, asynchronous, modem, and LAN adapters).

communications and systems management (C & SM). A set of tools, programs, and network functions used to plan, operate, and control an SNA communications network. C & SM runs on the store controller and must also exist at the host site.

compact disc- read-only memory (CD-ROM). (1) A 4.75-inch optical memory storage medium, capable of storing approximately 650 megabytes of data. Data is read optically by means of a laser. (2) A disc with information stored in the form of pits along a spiral track. The information is decoded by a compact-disc player and interpreted as digital audio data, which most computers can process.

compile. (1) To translate all or part of a program expressed in a high-level language into a computer program expressed in an intermediate language, an assembly language, or a machine language. (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler. (3) To translate a source program into an executable program (an object program). (4) To translate a program written in a high-level programming language into a machine language program.

compound files. Files that are kept on all store controllers.

computer architecture. The organizational structure of a computer system, including hardware and software.

concurrent conversations. The ability of a transaction program (TP) to manage more than one LU 6.2 conversation at the same time. When this ability is written into a TP, the TP is said to be *managing concurrent conversations*.

configuration. The group of devices, options, and programs that make up a data processing system or network as defined by the nature, number, and chief characteristics of its functional units. More specifically, the term may refer to a hardware configuration or a software configuration. See also *system configuration*.

configuration parameters. Variables in a configuration definition, the values of which characterize the relationship of a product, such as a bridge, to other products in the same network.

connect. In a LAN, to physically join a cable from a station to an access unit or network connection point. Contrast with *attach*.

contention. In a LAN, a situation in which two or more data stations are allowed by the protocol to start transmitting concurrently and thus risk collision.

contention loser. In APPC, the LU that must request and receive permission from the session partner LU to allocate a session. Contrast with *contention winner*.

contention winner. The LU that can allocate a session without requesting permission from the session partner LU. Contrast with *contention loser*.

contiguous. Touching or joining at the edge or boundary; adjacent. For example, an unbroken consecutive series of memory locations.

controller. A unit that controls input/output operations for one or more devices.

conversation. A logical connection between two programs over an LU type 6.2 session that allows them to communicate with each other while processing a transaction. See also *basic conversation* and *mapped conversation*.

conversation partner. One of the two programs involved in a conversation.

conversation state. The condition of a conversation that reflects what the past action on that conversation has been and that determines what the next set of actions may be.

corrective diskette (CD). A set of diskettes that contain modules to replace the modules in the active program subdirectory. The first diskette of the set must contain a product control file that describes which product the modules are to be applied to and a list of all modules that are to be replaced.

CRC. Cyclic redundancy check.

cursor. A movable point of light (or a short line) that indicates where the next character is to be entered on the display screen.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining data bases.

customer receipt. An itemized list of merchandise purchased and paid for by the customer.

customize. To tailor a program or store system through option selection.

cyclic redundancy check (CRC). Synonym for *frame check sequence (FCS)*.

D

data. (1) A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by human or automatic means. (2) Any representations such as characters or analog quantities to which meaning is or might be assigned.

data circuit-terminating equipment (DCE). In a data station, the equipment that provides the signal conversion and coding between the data terminal equipment (DTE) and the line.

data communication. (1) Transfer of information between functional units by means of data transmission according to a protocol. (2) The transmission, reception, and validation of data.

data file. A collection of related data records organized in a specific manner; for example, a payroll file (one record for each employee, showing such information as rate of pay and deductions) or an inventory file (one record for each inventory item, showing such information as cost, selling price, and number in stock.) See also *data set, file*.

data link. (1) Any physical link, such as a wire or a telephone circuit, that connects one or more remote terminals to a communication control unit, or connects one communication control unit with another. (2) The assembly of parts of two data terminal equipment (DTE) devices that are controlled by a link protocol, and the interconnecting data circuit, that enable data to be transferred from a data source to a data link. (3) In SNA, see also *link*. **Note:** A telecommunication line is only the physical medium of transmission. A data link includes the physical medium of transmission, the protocol, and associated devices and programs; it is both physical and logical.

data processing system. A network, including computer systems and associated personnel, that

accepts information, processes it according to a plan, and produces the appropriate results.

data rate. See *data transfer rate, line data rate*.

data set. Logically related records treated as a single unit. See also *file*.

data terminal equipment (DTE). (1) That part of a data station that serves as a data source, data receiver, or both. (2) Equipment that sends or receives data, or both.

data transfer. (1) The result of the transmission of data signals from any data source to a data receiver. (2) The movement, or copying, of data from one location and the storage of the data at another location.

data transfer rate. The average number of bits, characters, or blocks per unit of time passing between equipment in a data-transmission session. The rate is expressed in bits, characters, or blocks per second, minute, or hour.

data transmission. The conveying of data from one place for reception elsewhere by means of telecommunications.

dB. Decibel.

DBCS. Double-byte character set.

DCE. Data circuit-terminating equipment.

DDA. Data Distribution Application.

debug. To detect, diagnose, and eliminate errors in computer programs.

decibel (dB). (1) One tenth of a bel. (2) A unit that expresses the ratio of two power levels on a logarithmic scale. (3) A unit for measuring relative power. The number of decibels is 10 times the logarithm base (base 10) of the ratio of the measured power levels; if the measured levels are voltages (across the same or equal resistance), the number of decibels is twenty times the log of the ratio. See also *neper*.

default. Pertaining to an attribute, value, or option that is assumed when none is explicitly specified.

default value. The value the system supplies when the user does not specify a value.

delayed data maintenance. A function that allows the item record, the operator and the check authorization files to be maintained from the host on an immediate or a delayed basis.

destination. Any point or location, such as a node, station, or particular terminal, to which information is to be sent.

device. (1) A mechanical, electrical, or electronic contrivance with a specific purpose. (2) An input/output unit such as a terminal, display, or printer. See also *attaching device*.

device channel. In Toshiba Point-of-Sale terminals, a path along which signals for serial input/output devices can be sent. For these terminals, the device channel controller or adapter is contained on the system board.

diagnostic diskette. A diskette containing diagnostic modules or tests used by computer users and service personnel to diagnose hardware problems.

diagnostics. Modules or tests used by computer users and service personnel to diagnose hardware problems.

dialing. Using a dial or pushbutton telephone to initiate a telephone call. In telecommunication, attempting to establish a connection between a terminal and a telecommunication device over a switched line.

direct memory access (DMA). A procedure or method designed to transfer data between main storage and I/O units without intervention of the processing unit.

directory. (1) A table of identifiers and references that correspond to items of data. (2) An index that a control program uses to locate one or more blocks of data that are stored in separate areas of a data set in direct access storage.

disabled. (1) Pertaining to a state of a processing unit that prevents the occurrence of certain types of interruptions. (2) Pertaining to the state in which a transmission control unit or audio response unit cannot accept incoming calls on a line.

DISC. Disconnect character.

disk. A round, flat plate coated with a magnetic substance on which computer data is stored. See also *integrated disk*, *fixed disk*.

diskette. A thin, flexible magnetic disk permanently enclosed in a protective jacket. A diskette is used to store information for processing.

Disk Operating System (DOS). An operating system for computer systems that use disks and diskettes for auxiliary storage of programs and data.

display. (1) A visual presentation of data. (2) A device that presents visual information to the point-of-sale terminal operator and to the customer, or to the display station operator.

distributed. Physically separate but connected by cables.

Distributed Systems Executive (DSX). An IBM licensed program available for host systems that allows

the host system to get, send, and remove files, programs, formats and procedures in a network of computers.

DMA. Direct memory access

domain. An SSCP and the resources that it can control.

DOS. Disk Operating System.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with single-byte character set.

driver. Software component that controls a device.

drop. A cable that leads from a faceplate to the distribution panel in a wiring closet. When the Cabling System is used with the Token-Ring Network, a drop may form part of a lobe. See also *lobe*.

DSX. Distributed Systems Executive.

DTE. Data terminal equipment.

dump. (1) To write at a particular instant the contents of storage, or part of storage, onto another data medium for the purpose of safeguarding or debugging the data. (2) Data that has been dumped.

E

EAN. European article number.

EBCDIC. Extended binary-coded decimal interchange code.

EIA. Electronic Industries Association. See *EIA interface*.

EIA interface. An industry-accepted interface for connecting devices having voltage-related limits.

emulation. (1) The imitation of all or part of one computer system by another, primarily by hardware, so that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated computer system. (2) The use of programming techniques and special machine features to permit a computing system to execute programs written for another system.

enabled. (1) On a LAN, pertaining to an adapter or device that is active, operational, and able to receive frames from the network. (2) Pertaining to a state of a processing unit that allows the occurrence of certain

types of interruptions. (3) Pertaining to the state in which a transmission control unit or an audio response unit can accept incoming calls on a line.

envelope. (1) Information added to a frame or other message unit to allow it to be transmitted using a protocol other than the protocol in which the message unit originated. (2) To surround or enclose a message unit in information to allow the message unit to be transmitted using a protocol other than the protocol in which the message originated.

error condition. The condition that results from an attempt to use instructions or data that are invalid.

error message. A message that is issued because an error has been detected.

Ethernet. A 10-megabit baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and transmission. Ethernet uses carrier sense multiple access with collision detection (CSMA/CD).

European article number (EAN). A number that is assigned to and encoded on an article of merchandise for scanning in some countries.

evaluation. Reduction of an expression to a single value.

exchange identification (XID). The ID that is exchanged with the remote physical unit when an attachment is first established.

execute. To perform the actions specified by a program or a portion of a program.

execution. The process of carrying out an instruction or instructions of a computer program by a computer.

exit. To execute an instruction or statement within a portion of a program in order to terminate the execution of that portion. **Note:** Such portions of programs include loops, routines, subroutines, and modules.

expansion board. In a personal computer, a panel containing microchips that a user can install in an expansion slot to add memory or special features. Synonymous with *expansion card*, *extender card*.

expansion card. Synonym for *expansion board*.

expansion slot. In a personal computer, one of several receptacles in the system board of the system unit or expansion unit into which a user can install an expansion board such as a memory expansion option.

extended binary-coded decimal interchange code (EBCDIC). A coded character set consisting of 8-bit coded characters.

extender card. Synonym for *expansion board*.

F

fault. An accidental condition that causes a functional unit to fail to perform its required function.

feature. A part of a Toshiba product that may be ordered separately by the customer.

Feature Expansion. A card that plugs into a 4683 Point-of-Sale Terminal and allows additional devices to be used.

field. On a data medium or a storage medium, a specified area used for a particular category of data; for example, a group of character positions used to enter or display wage rates on a panel.

file. A named set of records stored or processed as a unit. For example, an invoice may form a record and the complete set of such records may form a file. See also *data file* and *data set*.

file name. (1) A name assigned or declared for a file. (2) The name used by a program to identify a file.

file server. (1) A store controller that maintains prime versions of all non-system mirrored files. (2) A high-capacity disk storage device or a computer that each computer on a network can access to retrieve files that can be shared among the attached computers.

file type. The attribute of a file that specifies to which store controllers it is distributed.

fixed disk (drive). In a personal computer system unit, a disk storage device that reads and writes on rigid magnetic disks. It is faster and has a larger storage capacity than a diskette and is permanently installed.

foreground. On a color display, the part of the display area that is the character itself.

foreground application. An interactive program that can be selected by system menus or started in command mode. Contrast with *background application*.

formatted diskette. A diskette on which track and sector control information has been written and that can be used by the computer to store data. **Note:** A diskette must be formatted before it can receive data.

frame. (1) The unit of transmission in some LANs, including the Token-Ring Network. It includes delimiters, control characters, information, and checking characters. On a token-ring network, a frame is created from a token when the token has data appended to it. On a token-bus network, all frames including the token frame contain a preamble, start delimiter, control address, optional data and checking characters, end delimiter, and are followed by a minimum silence period. (2) A housing for machine elements. (3) In synchronous

data link control (SDLC), the vehicle for every command, every response, and all information that is transmitted using SDLC procedures. Each frame begins and ends with a flag.

frame check sequence (FCS). (1) A system of error checking performed at both the sending and receiving station after a block-check character has been accumulated. (2) A numeric value derived from the bits in a message that is used to check for any bit errors in transmission. (3) A redundancy check in which the check key is generated by a cyclic algorithm. Synonymous with *cyclic redundancy check (CRC)*.

franking. Printing an indication on a document that the document has been processed. This franking may be a store header line, a "total" line, or a transaction number that is printed when a check, a discount coupon, or a gift certificate is inserted in the document insert station of the point-of-sale terminal during certain types of transactions.

frequency. The rate of signal oscillation, expressed in hertz (cycles per second).

function. (1) A specific purpose of an entity, or its characteristic action. (2) A subroutine that returns the value of a single variable. (3) In data communications, a machine action such as a carriage return or line feed.

G

gateway. A device and its associated software that interconnect networks of systems of different architectures. The connection is usually made above the Reference Model network layer. For example, a gateway allows LANs access to System/370 host computers. Contrast with *bridge* and *router*.

group. (1) A set of related records that have the same value for a particular field in all records. (2) A collection of users who can share access authorities for protected resources. (3) A list of names that are known together by a single name.

H

hardware. Physical equipment as opposed to programs, procedures, rules, and associated documentation.

HCP. Host command processor for advanced data communications.

HCP. Host command processor.

header. The portion of a message that contains control information for the message such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

host application program. An application program that the host processor executes.

host command processor (HCP). The SNA logical unit of the programmable Store System store controller.

host computer. (1) The primary or controlling computer in a multi-computer installation or network. (2) In a network, a processing unit in which resides a network access method. Synonymous with *host processor*.

host processor. (1) In a network, a computer that primarily provides services such as computation, data base access, or special programs or programming languages. (2) Synonym for *host computer*.

host processor. (1) A processor that controls all or part of a user application network. (2) In a network, the processing unit in which resides the access method for the network. (3) In an SNA network, the processing unit that contains a system services control point (SSCP). (4) A processing unit that executes the access method for attached communication controllers. (5) The processing unit required to create and maintain PSS. Synonymous with *host computer*.

I

IBM Disk Operating System (DOS). A disk operating system based on MS-DOS**.

identifier. String of characters used to name elements of a program, such as variable names, reserved words, and user-defined function names.

idles. Signals sent along a ring network when neither frames nor tokens are being transmitted.

image version. Copy of a prime version of a file. See *prime version*.

inactive. (1) Not operational. (2) Pertaining to a node or device not connected or not available for connection to another node or device. (3) In the Token-Ring Network, pertaining to a station that is only repeating frames or tokens, or both.

information (I) frame. A frame in I format used for numbered information transfer. See also *supervisory frame*, *unnumbered frame*.

initialize. In a LAN, to prepare the adapter (and adapter support code, if used) for use by an application program.

initial program load (IPL). The initialization procedure that causes an operating system to begin operation.

input device. Synonym for *input unit*.

input field. An unprotected display field that the terminal operator can add to, modify, or erase by using the keyboard. Contrast with *protected field*.

input/output (I/O). (1) Pertaining to a device whose parts can perform an input process and an output process at the same time. (2) Pertaining to a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process.

input unit. A device in a data processing system by means of which data can be entered into the system. Synonymous with *input device*.

insert. To make an attaching device an active part of a LAN.

integrated. Arranged together as one unit.

integrated disk. An integral part of the processor that is used for magnetically storing files, application programs, and diagnostics. Synonymous with *disk*.

interactive. Pertaining to an application or program in which each entry calls forth a response from a system or program. An interactive program may also be conversational, implying a continuous dialog between the user and the system.

interface. (1) A shared boundary between two functional units, defined by functional characteristics, common physical interconnection characteristics, signal characteristics, and other characteristics as appropriate. (2) A shared boundary. An interface may be a hardware component to link two devices or a portion of storage or registers accessed by two or more computer programs. (3) Hardware, software, or both, that links systems, programs, or devices.

interference. (1) The prevention of clear reception of broadcast signals. (2) The distorted portion of a received signal.

interleave. To insert segments of one program into another program so that the two programs can, in effect, be executed at the same time.

interrupt. (1) A suspension of a process, such as execution of a computer program, caused by an external event and performed in such a way that the process can be resumed. (2) To stop a process in such a way that it can be resumed. (3) In data communication, to take an action at a receiving station that causes the sending station to end a transmission. (4) A means of passing processing control from one software or microcode module or routine to another, or of requesting a particular software, microcode, or hardware function.

interrupt level. The means of identifying the source of an interrupt, the function requested by an interrupt, or the code or feature that provides a function or service.

I/O. Input/output.

I/O device. Equipment for entering and receiving data from the system.

IP. Internet Protocol.

IPL. Initial program load.

isochronous. Time-dependent. Refers to processes in which data must be delivered within certain time constraints.

item. (1) One member of a group. (2) In a store, one unit of a commodity, such as one box, one bag, or one can. Usually an item is the smallest unit of a commodity to be sold.

J

Java. An object-oriented programming language designed to be platform independent.

Java application. A Java Virtual Machine (JVM) combined with its class and parameters.

Java Virtual Machine (JVM). Java interpreter that runs the class.

jumper. A connector between two pins on a network adapter that enables or disables an adapter option, feature, or parameter value.

JUCC. Japan Unified Cash Card.

JVM. See Java Virtual Machine.

K

K. When referring to storage capacity, a symbol that represents two to the tenth power, or 1024.

Kb. Kilobit.

KB. Kilobyte.

keyboard. A group of numeric keys, alphabetic keys, special character keys, or function keys used for entering information into the terminal and into the system.

keyed file. Type of file composed of keyed records. Each keyed record has two parts: a key and data. A key is used to identify and access each record in the file.

kilobit (Kb). 1024 binary digits.

kilobyte (KB). 1024 bytes for processor and data storage (memory) size.

L

label. Constant, either numeric or literal, that references a statement or function.

LAN. Local area network.

LAN segment. (1) Any portion of a LAN (for example, a single bus or ring) that can operate independently but is connected to other parts of the establishment network by bridges. (2) An entire ring or bus network without bridges. See *cable segment*, *ring segment*.

LCD. Liquid crystal display.

leased line. Synonym for *nonswitched line*.

LED. Light-emitting diode.

light-emitting diode (LED). A semiconductor chip that gives off visible or infrared light when activated.

line connection. In the Toshiba Store System, the physical connection (or equipment) between nodes that provides two-way communication and error correction and detection between one link station and one or more other link stations. **Note:** In SNA, this physical connection is called a *link connection*. In the Toshiba Store System, it is called a *line connection*.

line data rate. The rate of data transmission over a telecommunications link.

link. (1) In the Toshiba Store System, the logical connection between nodes including the end-to-end link control procedures. (2) The combination of physical media, protocols, and programming that connects devices on a network. (3) In computer programming, the part of a program, in some cases a single instruction or an address, that passes control and parameters between separate portions of the computer program. (4) To interconnect items of data or portions of one or more computer programs. (5) In SNA, the combination of the link connection and link stations joining network nodes. See also *link connection*. **Note:** A link connection is the physical medium of transmission; for example, a telephone wire or a microwave beam. A link includes the physical medium of transmission, the protocol, and associated devices and programming; it is both logical and physical.

link connection. (1) All physical components and protocol machines that lie between the communicating link stations of a link. The link connection may include a switched or leased physical data circuit, a LAN, or an X.25 virtual circuit. (2) In SNA, the physical equipment providing two-way communication and error correction and detection between one link station and one or more other link stations. (3) In the Store System, the logical link providing two-way communication of data from one network node to one or more other network nodes.

listing. A printout of source code.

load. In computer programming, to enter data into memory or working registers.

lobe. In the Token-Ring Network, the section of cable (which may consist of several segments) that connects an attaching device to an access unit.

local area network (LAN). A computer network located on a user's premises within a limited geographical area. **Note:** Communication within a LAN is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation.

local program. The program being discussed within a particular context. Contrast with *remote program*.

logical file name (LFN). An abbreviated file name used to represent either an entire file name or the drive and subdirectory path part of the file name.

logical link. In an MVS/VS multisystem environment, the means by which a physical link is related to the transactions and terminals that can use the physical link.

logical unit (LU). (1) In SNA, a port through which an end user accesses the SNA network in order to communicate with another end user and through which the end user accesses the functions provided by system services control points (SSCPs). An LU can support at least two sessions, one with an SSCP and one with another LU, and may be capable of supporting many sessions with other logical units. (2) A type of network addressable unit that enables end users to communicate with each other and gain access to network resources.

logon (n). The procedure for starting up a point-of-sale terminal or store controller for normal sales operations by sequentially entering the correct security number and transaction number. Synonymous with *sign-on*.

log on (v). (1) To initiate a session. (2) In SNA products, to initiate a session between an application program and a logical unit (LU). Synonymous with *sign-on*.

loop. (1) A set of instructions that may be executed repeatedly while a certain condition prevails. See also *store loop*. (2) A closed unidirectional signal path connecting input/output devices to a network.

LU. Logical unit.

M

magnetic stripe. The magnetic material (similar to recording tape) on merchandise tickets, credit cards, and employee badges. Information is recorded on the

stripe for later “reading” by the magnetic stripe reader (MSR) or magnetic wand reader attached to the point-of-sale terminal.

magnetic stripe reader (MSR). A device that reads coded information from a magnetic stripe on a card, such as a credit card, as it passes through a slot in the reader.

maintenance analysis procedure (MAP). Deprecated term for *procedure*. See *procedure*.

maintenance diskette. See *corrective diskette*.

Manufacturing Automated Protocol (MAP). A broadband LAN with a bus topology that passes tokens from adapter to adapter on a coaxial cable.

MAP. (1) Maintenance analysis procedure. (2) Manufacturing Automated Protocol.

mapped conversation. A conversation in which programs exchange data records with arbitrary data formats agreed upon by the applications programmers.

mapping. Establishing a correspondence between the elements of one set and the elements of another set.

master store controller. The store controller that maintains prime versions of system mirrored files and all compound files.

Mb. Megabit.

MB. Megabyte.

MCF Network. Multiple store controllers communicating on a network using DDA. This provides data redundancy among the store controllers.

media. Plural form of *medium*.

medialess. Not fitted with a direct access storage device, such as a diskette drive or fixed disk drive, as in some models of Toshiba Point of Sale Terminals.

medium. (1) A physical carrier of electrical or optical energy. (2) A physical material in or on which data may be represented.

megabit (Mb). A unit of measure for throughput. 1 megabit = 1,048,576 bits.

megabyte (MB). A unit of measure for data. 1 megabyte = 1,048,576 bytes.

megahertz (MHz). A unit of measure of frequency. 1 megahertz = 1,000,000 hertz.

memory. Program-addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing.

message. (1) An arbitrary amount of information whose beginning and end are defined or implied. (2) A

group of characters and control bit sequences transferred as an entity. (3) In telecommunication, a combination of characters and symbols transmitted from one point to another. (4) A logical partition of the user device’s data stream to and from the adapter. See also *error message*, *operator message*.

MHz. Megahertz.

Micro Channel. The architecture used by Personal System/2 computers, Models 50 and above. This term is used to distinguish these computers from personal computers using a PC I/O channel, such as an IBM PC, XT, or a Personal System/2 computer, Model 25 or 30.

migration. Upgrade of a program to a newer version or release.

mirrored files. Files that are kept on both the Master Store Controller and the Alternate Master Store Controller or on both the File Server and Alternate File Server. System mirrored files are kept on the Master Store Controller and Alternate Store Controller and non-system mirrored files are kept on the File Server and Alternate File Server.

Mod1. A generic name used to refer to a point-of-sale terminal in the 4690 Store System that loads and executes programs. A Mod1 can be any of the following models: 4683-001, 4683-A01, 4683-P11, 4683-P21, 4683-P41, 4683-421, 4693-xx1, and 4694-xx4 (terminal part if a controller/terminal).

Mod2. A generic name used to refer to a point-of-sale terminal in the 4690 Store System that does not load and execute programs, but attaches to a terminal that does. A Mod2 can be one of the following models: 4683-002, 4683-A02, or 4693-2x2.

modem (MODulator/DEModulator). A device that converts digital data from a computer to an analog signal that can be transmitted in a telecommunication line, and converts the analog signal received to data for the computer.

module. A program unit that is discrete and identifiable with respect to compiling, combining with other units, and load; for example, the input to, or output from, an assembler, compiler, linkage editor, or executive routine.

modulo check. A function designed to detect most common input errors by performing a calculation on values entered into a system by an operator or scanning device.

monitor. (1) A functional unit that observes and records selected activities for analysis within a data processing system. Possible uses are to show significant departures from the norm, or to determine levels of utilization of particular functional units. (2) Software or hardware that observes, supervises, controls, or verifies operations of a system.

monochrome display. A display device that presents display images in only one color.

MSR. Magnetic stripe reader.

multiple controller system. Synonym for *MCF Network*.

multipoint. Pertaining to communication among more than two stations over a single telecommunication line.

multipoint line. A telecommunication line or circuit connecting two or more stations. Contrast with *point-to-point line*.

N

name. An alphanumeric term that identifies a data set, statement, program, or cataloged procedure.

n-bit byte. A string that consists of n bits.

NCP. Network control program.

neper. A unit for measuring power. The number of nepers is the logarithm (base e) of the ratio of the measured power level.

NetBIOS. Network Basic Input/Output System.

NetView. A host-based IBM network management licensed program that provides communication network management (CNM) or communications and systems management (C & SM) services.

NetView Distribution Manager (NetView DM). A component of the NetView family supporting resource distribution within *Change Management*, and providing central control of software and microcode distribution and installation, to processors in a distributed/departmental (SNA) network system. It allows a similar control of user data objects across the network, and provides the facilities to support the remote initiation of command lists.

network. (1) A configuration of data processing devices and software connected for information interchange. (2) An arrangement of nodes and connecting branches. Connections are made between data stations.

network administrator. A person who manages the use and maintenance of a network.

network architecture. The logical structure and operating principles of a computer network. See also *systems network architecture (SNA)* and *Open Systems Interconnect (OSI) architecture*. **Note:** The operating principles of a network include those of services, functions, and protocols.

Network Basic Input/Output System (NetBIOS). A message interface used on LANs to provide message,

print server, and file server functions. The NetBIOS application program interface (API) provides a programming interface to the LAN so that an application program can have LAN communication without knowledge and responsibility of the data link control (DLC) interface.

network control program (NCP). A control program for the 3704 or 3705 Communications Controller, generated by the user from a library of Toshiba-supplied modules.

network file system (NFS). A system that allows you to mount remote file systems across homogeneous and heterogeneous systems.

network management vector transport (NMVT). The portion of an alert transport frame that contains the alert message.

NFS. network file system

node. (1) Any device, attached to a network, that transmits and/or receives data. (2) An end point of a link, or a junction common to two or more links in a network. Nodes can be processors, controllers, or workstations. Nodes can vary in routing and other functional capabilities. (3) In a network, a point where one or more functional units interconnect transmission lines.

node address. The address of an adapter on a LAN.

nonswitched line. (1) A connection between systems or devices that does not have to be made by dialing. Contrast with *switched line*. (2) A telecommunication line on which connection does not have to be established by dialing. Synonymous with *leased line*.

nonvolatile random access memory (NVRAM). Random access memory that retains its contents after electrical power is shut off.

NRZI. (1) Non-return-to-zero inverted transmission. (2) Non-return-to-reference transmission in which the zeros are represented by a bit cell boundary transition in the information signal, and ones are represented by the absence of a bit cell boundary transition.

NVRAM. nonvolatile random access memory

O

OCR. Optical character recognition.

offline. Operation of a functional unit without the control of a computer or control unit.

online. Operation of a functional unit that is under the continual control of a computer or control unit. The term also describes a user's access to a computer using a terminal.

open. (1) To make an adapter ready for use. (2) A break in an electrical circuit. (3) To make a file ready for use.

Open Systems Interconnect (OSI). (1) The interconnection of open systems in accordance with specific ISO standards. (2) The use of standardized procedures to enable the interconnection of data processing systems. **Note:** OSI architecture establishes a framework for coordinating the development of current and future standards for the interconnection of computer systems. Network functions are divided into seven layers. Each layer represents a group of related data processing and communication functions that can be carried out in a standard way to support different applications.

Open Systems Interconnect (OSI) architecture. Network architecture that adheres to a particular set of ISO standards that relates to Open Systems Interconnect (OSI).

Open Systems Interconnect (OSI) Reference Model. A model that represents the hierarchical arrangement of the seven layers described by the Open Systems Interconnect (OSI) architecture.

operating system. Software that controls the execution of programs. An operating system may provide services such as resource allocation, scheduling, input/output control, and data management. Examples are DOS and OS/2.

Operating System/2 (OS/2). A set of programs that control the operation of high-speed large-memory IBM Personal Computers (such as the Personal System/2 computer, Models 50 and above), providing multitasking and the ability to address up to 16 MB of memory. Contrast with *Disk Operating System (DOS)*.

operation. (1) A defined action, namely, the act of obtaining a result from one or more operands in accordance with a rule that completely specifies the result for any permissible combination of operands. (2) A program step undertaken or executed by a computer. (3) An action performed on one or more data items, such as adding, multiplying, comparing, or moving.

operational environment. (1) A summation of all of the Toshiba-supplied basic functions and the user programs that can be executed by the store controller to enable the devices in the system to perform specific operations. (2) The collection of Toshiba-supplied controller data and user programs, plus lists, tables, control blocks, and files that reside in a controller and control its operation. (3) The physical environment (for example: temperature, humidity, layout, or power requirements) that is needed for proper machine performance.

operator. (1) A symbol that represents the action being performed in a mathematical operation. (2) A person who operates a machine.

operator message. A message from the operating system or a program telling the operator to perform a specific function or informing the operator of a specific condition within the system, such as an error condition.

optical character recognition (OCR). The machine identification of printed characters through the use of light-sensitive devices.

option. (1) A specification in a statement, a selection from a menu, or a setting of a switch, that may be used to influence the execution of a program. (2) A hardware or software function that may be selected or enabled as part of a configuration process. (3) A piece of hardware (such as a network adapter) that can be installed in a device to modify or enhance device function.

OS. Operating system.

OS/2. Operating System/2.

OSI. Open Systems Interconnect.

OS/VS. Operating System/Virtual Storage.

owner. In relation to files, an owner is the user that creates the file and therefore has complete access to the file.

P

pacing. A technique by which a receiving component controls the rate of transmission by a sending component to prevent overrun or congestion.

packet. (1) In data communication, a sequence of binary digits, including data and control signals, that is transmitted and switched as a composite whole. (2) Synonymous with *data frame*. Contrast with *frame*.

packet assembler/disassembler (PAD). A functional unit that enables data terminal equipments (DTEs) not equipped for packet switching to access a packet switched network.

packing. Method of conserving disk storage space by stripping the high-order nibbles from ASCII numerals and storing the remaining low-order nibbles two to a byte.

PAD. Packet assembler/disassembler.

page. (1) The portion of a panel that is shown on a display surface at one time. (2) To move back and forth among the pages of a multiple-page panel. See also *scroll*. (3) In a virtual storage system, a fixed-length block that has a virtual address and is transferred as a unit between main storage and auxiliary storage.

panel. The complete set of formatted information that appears in a single display on a visual display unit.

parallel port. (1) A port that transmits the bits of a byte in parallel along the lines of the bus, one byte at a time, to an I/O device. (2) On a personal computer, it is used to connect a device that uses a parallel interface, such as a dot matrix printer, to the computer. Contrast with *serial port*.

parameter. (1) A name in a procedure that is used to refer to an argument passed to that procedure. (2) A variable that is given a constant value for a specified application and that may denote the application. (3) An item in a menu or for which the user specifies a value or for which the system provides a value when the menu is interpreted. (4) Data passed between programs or procedures.

parity (even). A condition when the sum of all of the digits in an array of binary digits is even.

parity (odd). A condition when the sum of all of the digits in an array of binary digits is odd.

partner. See *conversation partner*.

partner terminal. The term used to describe the relationship of a Mod 1 terminal and Mod 2 terminal when they are attached to each other.

password. In computer security, a string of characters known to the computer system and a user, who must specify it to gain full or limited access to a system and to the data stored within it.

path. (1) Reference that specifies the location of a particular file within the various directories and subdirectories of a hierarchical file system. (2) In a network, any route between any two nodes. (3) The route traversed by the information exchanged between two attaching devices in a network. (4) A command in DOS and OS/2 that specifies directories to be searched for commands or batch files that are not found by a search of the current directory.

PCI DSS. Payment Card Industry Data Security Standards.

peer node. Any *other* SNA type (2.1) node (another 4680/4690 store controller, AS/400, or others).

permanent virtual circuit (PVC). A virtual circuit that has a logical channel permanently assigned to it at each data terminal equipment (DTE). A call establishment protocol is not required.

personal computer (PC). A desk-top, free-standing, or portable microcomputer that usually consists of a system unit, a display, a keyboard, one or more diskette drives, internal fixed-disk storage, and an optional printer. PCs are designed primarily to give independent computing power to a single user and are inexpensively priced for purchase by individuals or small businesses.

Examples include the various models of the IBM Personal Computers, and the Personal System/2 computer.

personal identification number (PIN). A numeric identification code assigned to a customer to protect funds and data from unauthorized users.

physical link. In an MVS/VS multisystem environment, the actual hardware connection between two systems. Contrast with *logical link*.

physical unit (PU). In SNA, the component that manages and monitors the resources of a node, such as attached links and adjacent link stations, as requested by a system services control point (SSCP) through an SSCP-SSCP session.

pipe. A sequential file in a memory buffer that is used to pass messages from one program to another.

PLD. Power line disturbance.

plug. (1) A connector for attaching wires from a device to a cable, such as a store loop. A plug is inserted into a receptacle or plug. (2) To insert a connector into a receptacle or socket.

point-of-sale terminal. A unit that provides point-of-sale transaction, data collection, credit authorization, price look-up, and other inquiry and data entry functions.

point-to-point line. A switched or nonswitched telecommunication line that connects a single remote station to a computer. Contrast with *multipoint line*.

polling. (1) Interrogation of devices for purposes such as to avoid contention, to determine operational status, or to determine readiness to send or receive data. (2) In data communication, the process of inviting data stations to transmit, one at a time. The polling process usually involves the sequential interrogation of several data stations.

polling characters (address). A set of characters specific to a terminal and the polling operation; response to these characters indicates to the computer whether the terminal has a message to enter.

port. (1) An access point for data entry or exit. (2) A connector on a device to which cables for other devices such as display stations and printers are attached. Synonymous with *socket*.

post. (1) To affix to a usual place. (2) To provide items such as return code at the end of a command or function. (3) To define an appendage routine. (4) To note the occurrence of an event.

POST. Power-On Self Test.

power line disturbance (PLD). Interruption or reduction of electrical power.

Power-On Self Test (POST). A series of diagnostic tests that are run automatically each time the computer's power is switched on.

presentation space (PS). In 3270 emulation, the image of the 3270 screen data that is held in random access memory. This screen appears on the store controller or the terminal display when 3270 emulation is used in operator console mode; it is the virtual screen for applications using the 3270 emulator API. The presentation space is fixed as 24 lines of 80 characters on the display.

primary adapter. In a personal computer that is used on a LAN and that supports installation of two network adapters, the adapter that uses standard (or default) mapping between adapter shared RAM, adapter ROM, and designated computer memory segments. The primary adapter is usually designated as adapter 0 in configuration parameters. Contrast with *alternate adapter*.

primary application. A program that controls the normal operating environment of your store (for example, programs that provide sales support).

primary store controller. The store controller designated to control the store loop under normal conditions.

prime version. The version of a file to which updates are made. The prime version of a file may be maintained on either the Master Store Controller or the File Server. Copies of the prime version, called image versions, are distributed to other store controllers.

printout. Any printed document produced by a point-of-sale terminal printer or by some other printer.

problem determination. The process of determining the source of a problem as being a program component, a machine failure, a change in the environment, a common-carrier link, a user-supplied device, or a user error.

procedure. (1) A set of related control statements that cause one or more programs to be performed. (2) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a procedure call. (3) A set of instructions that gives a service representative a step-by-step procedure for tracing a symptom to the cause of failure.

processor. In a computer, a functional unit that interprets and executes instructions.

Programmable Store System (PSS). A store system, such as the Toshiba Store System, that can be programmed to perform user-determined functions.

prompt. A character or word displayed by the operating system to indicate that it is ready to accept input.

protected field. A display field that the terminal operator cannot add to, modify, or erase using the keyboard. Contrast with *input field* and *unprotected field*.

protocol. (1) A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (2) In SNA, the meanings of and the sequencing rules for requests and responses used for managing the network, transferring data, and synchronizing the states of network components. (3) A specification for the format and relative timing of information exchanged between communicating parties.

PS. Presentation space.

PSS. Programmable Store System.

PU. Physical unit.

public switched (telephone) network (PSN). A telephone network that provides lines and exchanges to the public. It is operated by the communication common carriers in the USA and Canada, and by the PTT Administrations in other countries.

PVC. Permanent virtual circuit.

Q

queue. A line or list formed by items in a system waiting for service; for example, tasks to be performed or messages to be transmitted in a message routing system.

R

RAM. Random access memory.

RAM disk. Synonym for *virtual drive*.

RAM paging. A technique that allows the computer software to access all of the RAM on adapters that contain 64 KB of RAM, without having to map the entire shared RAM into the computer's memory map. The shared RAM on the adapter is paged into the computer's memory map one 16 KB page at a time.

random access. An access mode in which specific logical records are obtained from or placed into a mass storage file in a nonsequential manner.

random access memory (RAM). A computer's or adapter's volatile storage area into which data may be entered and retrieved in a nonsequential manner.

RCMS. Remote change management server.

read. To acquire or to interpret data from a storage device, from a data medium, or from another source.

read-only memory (ROM). A computer's or adapter's storage area whose contents cannot be modified by the user except under special circumstances.

real storage. The main storage in an virtual storage system. Contrast with *virtual storage (VS)*.

receive. To obtain and store information transmitted from a device.

record. A collection of related items of data, treated as a unit; for example, in stock control, each invoice could constitute one record. A complete set of such records may form a file.

reference diskette. A diskette shipped with the point-of-sale equipment. The diskette contains code and files used for configuration of options and for hardware diagnostic testing.

remote change management server (RCMS). The Toshiba Store System function that interfaces with the host DSX program for file transmission.

remote program. The program at the other end of a conversation with respect to the reference program. Contrast with *local program*.

remote program load (RPL). A function provided by adapter hardware components and software that enables one computer to load programs and operating systems into the memory of another computer, without requiring the use of a diskette or fixed disk at the receiving computer.

remove. (1) To take an attaching device off a network. (2) To stop an adapter from participating in data passing on a network.

response. The information the network control program sends to the access method, usually in answer to a request received from the access method. (Some responses, however, result from conditions occurring within the network control program, such as accumulation of error statistics.)

retry. In data communication, sending the current block of data a prescribed number of times or until it is entered correctly and accepted.

return code. (1) A value (usually hexadecimal) provided by an adapter or a program to indicate the result of an action, command, or operation. (2) A code used to influence the execution of succeeding instructions. (3) A value established by the programmer to be used to influence subsequent program action. This value can be printed as output or loaded in a register.

ring network. A network configuration in which a series of attaching devices is connected by unidirectional transmission links to form a closed path. A ring of a Token-Ring Network is referred to as a LAN segment or as a token-ring network segment.

ring segment. Any section of a ring that can be isolated (by unplugging connectors) from the rest of the ring. A segment can consist of a single lobe, the cable between access units, or a combination of cables, lobes, and/or access units. See *cable segment*, *LAN segment*.

ring station. A station that supports the functions necessary for connecting to the LAN and for operating with the token-ring protocols. These include token handling, transferring copied frames from the ring to the using node's storage, maintaining error counters, observing medium access control (MAC) sublayer protocols (for address acquisition, error reporting, or other duties), and (in the full-function native mode) directing frames to the correct data link control (DLC) link station.

ring status. The condition of the ring.

RIPL. Remote IPL.

RMA. Remote Management Agent.

ROM. Read-only memory.

root directory. Highest or base level directory in a hierarchical file system. Subdirectories branch off of the root directory.

router. An attaching device that connects two LAN segments, which use similar or different architectures, at the Reference Model network layer. Contrast with *bridge* and *gateway*.

routing. (1) The assignment of the path by which a message will reach its destination. (2) The forwarding of a message unit along a particular path through a network, as determined by the parameters carried in the message unit, such as the destination network address in a transmission header.

RPL. Remote program load.

S

SAA. Systems Application Architecture.

SABM. Set asynchronous balanced mode.

satellite. (1) A computer that is under the control of another computer and performs subsidiary operations. (2) An offline auxiliary computer. (3) A point-of-sale terminal under the control of a master terminal.

SBCS. Single-byte character set.

scan. To pass an item over or through the scanner so that the encoded information is read. See also *wandering*.

scanner. A device that examines the bar code on merchandise tickets, credit cards, and employee badges and generates analog or digital signals corresponding to the bar code.

scroll. To move all or part of the display image vertically or horizontally to display data that cannot be observed within a single display image. See also *page (2)*.

SDLC. Synchronous Data Link Control.

SDLC link. A data link over which communications are conducted using the Synchronous Data Link Control (SDLC) discipline.

secondary application. A user-written program that is designed to operate with operator intervention.

sector. A 512-byte area of the control unit diskette, the amount of data that is transferred at one time to or from the diskette.

segment. See *cable segment, LAN segment, ring segment*.

serial port. On personal computers, a port used to attach devices such as display devices, letter-quality printers, modems, plotters, and pointing devices such as light pens and mice; it transmits data one bit at a time. Contrast with *parallel port*.

server. (1) A device, program, or code module on a network dedicated to providing a specific service to a network. (2) On a LAN, a data station that provides facilities to other data stations. Examples are a file server, print server, and mail server.

session. (1) A connection between two application programs that allows them to communicate. (2) In SNA, a logical connection between two network addressable units that can be activated, tailored to provide various protocols, and deactivated as requested. (3) The data transport connection resulting from a call or link between two devices. (4) The period of time during which a user of a node can communicate with an interactive system, usually the elapsed time between log on and log off. (5) In network architecture, an association of facilities necessary for establishing, maintaining, and releasing connections for communication between stations.

session group. In System/36 advanced program-to-program communication, a number of sessions managed as a unit.

set asynchronous balanced mode (SABM). In communications, a data link control command used to establish a data link connection with the destination in asynchronous balanced mode. See also *asynchronous balanced mode (ABM)*.

shared RAM. Random access memory on an adapter that is shared by the computer in which the adapter is installed.

signal. (1) A time-dependent value attached to a physical phenomenon for conveying data. (2) A variation of a physical quantity, used to convey data.

sign-on. (1) A procedure to be followed at a terminal or workstation to establish a link to a computer. (2) To begin a session at a workstation.

single-byte character set (SBCS). A character set in which each character is represented by a one-byte code. Contrast with double-byte character set.

SNA. Systems Network Architecture.

socket. Synonym for *port (2)*.

source. The origin of any data involved in a data transfer.

SSCP. System services control point.

state. See *conversation state*.

station. (1) A point-of-sale terminal that consists of a processing unit, a keyboard, and a display. It can also have input/output devices, such as a printer, a magnetic stripe reader or cash drawers. (2) A communication device attached to a network. The term used most often in LANs is an *attaching device* or *workstation*. (3) An input or output point of a system that uses telecommunication facilities; for example, one or more systems, computers, terminals, devices, and associated programs at a particular location that can send or receive data over a telecommunication line. See also *attaching device, workstation*.

store controller. A programmable unit in a network used to collect data, to direct inquiries, and to control communication within a point-of-sale system.

store loop. In the Store System, a cable over which data is transmitted between the store controller and the point-of-sale terminals.

Store Loop Adapter. A hardware component used to connect the loop to a store controller.

subarea node. An SNA type 5 node (a host processor), which will control all communications with the store controller.

subdirectory. Any level of file directory lower than the root directory within a hierarchical file system.

subordinate store controller. A store controller that receives copies of all system compound files and may also receive all application compound files.

supervisory (S) frame. A frame in supervisory format used to transfer supervisory control functions. See also *information frame*, *unnumbered frame*.

SVC. Switched virtual circuit.

switch. On an adapter, a mechanism used to select a value for, enable, or disable a configurable option or feature.

switched line. A telecommunication line in which the connection is established by dialing. Contrast with *nonswitched line*.

switched virtual circuit (SVC). A virtual circuit that is requested by a virtual call. It is released when the virtual circuit is cleared.

symbolic destination name. Variable corresponding to an entry in the side information.

synchronous. (1) Pertaining to two or more processes that depend upon the occurrence of a specific event such as a common timing signal. (2) Occurring with a regular or predictable timing relationship.

Synchronous Data Link Control (SDLC). A discipline conforming to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute (ANSI) and High-level Data Link Control (HDLC) of the International Organization for Standardization, for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges may be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection may be point-to-point, multipoint, or loop.

system. In data processing, a collection of people, machines, and methods organized to accomplish a set of specific functions. See also *data processing system* and *operating system*.

system board. In a system unit, the main circuit board that supports a variety of basic system devices, such as a keyboard or a mouse, and provides other basic system functions.

system configuration. A process that specifies the devices and programs that form a particular data processing system.

Systems Application Architecture (SAA). An architecture developed by IBM that consists of a set of selected software interfaces, conventions, and protocols, and that serves as a common framework for application development, portability, and use across different Toshiba hardware systems.

system services control point (SSCP). In SNA, the focal point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory

support and other session services for end users of the network. Multiple SSCPs, cooperating as peers, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units (PUs) and logical units (LUs) within its domain.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. **Note:** The layered structure of SNA allows the ultimate origins and destinations of information, that is, the end users, to be independent of, and unaffected by, the specific SNA network services and facilities used for information exchange.

T

task. A basic unit of work.

TCC Network. A system in which the terminals and controllers communicate using either a store loop, a token-ring or an Ethernet.

telephone twisted pair. One or more twisted pairs of copper wire in the unshielded voice-grade cable commonly used to connect a telephone to its wall jack. Also referred to as “unshielded twisted pair” (UTP).

tender. Money, checks, coupons, or trading stamps used as payment for merchandise or service.

terminal. In data communication, a device, usually equipped with a keyboard and a display, capable of sending and receiving information over a communication channel.

terminal number. A number assigned to a terminal to identify it for addressing purposes.

threshold. (1) A level, point, or value above which something is true or will take place and below which it is not true or will not take place. (2) In IBM bridge programs, a value set for the maximum number of frames that are not forwarded across a bridge due to errors, before a “threshold exceeded” occurrence is counted and indicated to network management programs. (3) An initial value from which a counter is decremented from an initial value. When the counter reaches zero or the threshold value, a decision is made and/or an event occurs.

till. A tray in the cash drawer of the point-of-sale terminal, used to keep the different denominations of bills and coins separated and easily accessible.

token. A sequence of bits passed from one device to another on the token-ring network that signifies permission to transmit over the network. It consists of a starting delimiter, an access control field, and an end delimiter. The frame control field contains a token bit

that indicates to a receiving device that the token is ready to accept information. If a device has data to send along the network, it appends the data to the token. When data is appended, the token then becomes a frame. See *frame*.

token-ring. A network with a ring topology that passes tokens from one attaching device (node) to another. A node that is ready to send can capture a token and insert data for transmission.

token-ring network. (1) A ring network that allows unidirectional data transmission between data stations by a token-passing procedure over one transmission medium so that the transmitted data returns to and is removed by the transmitting station. The Token-Ring Network is a baseband LAN with a star-wired ring topology that passes tokens from network adapter to network adapter. (2) A network that uses a ring topology, in which tokens are passed in a circuit from node to node. A node that is ready to send can capture the token and insert data for transmission. (3) A group of interconnected token-rings.

TP. Transaction program.

trace. (1) A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed. (2) A record of the frames and bytes transmitted on a network.

transaction. (1) The process of recording item sales, processing refunds, recording coupons, handling voids, verifying checks before tendering, and arriving at the amount to be paid by or to a customer. The receiving of payment for merchandise or service is also included in a transaction. (2) In an SNA network, an exchange between two programs that usually involves a specific set of initial input data that causes the execution of a specific task or job. Examples of transactions include the entry of a customer's deposit that results in the updating of the customer's balance, and the transfer of a message to one or more destination points.

transaction program (TP). A program that processes transactions in or through a logical unit (LU) type 6.2 in an SNA network. Application transaction programs are end users in an SNA network; they process transactions for service transaction programs and for other end users. Service transaction programs are Toshiba-supplied programs that typically provide utility services to application transaction programs.

transmission. The sending of data from one place for reception elsewhere.

transmit. To send information from one place for reception elsewhere.

twisted pair. A transmission medium that consists of two insulated conductors twisted together to reduce noise.

typematic. A keyboard button that will continue to enter characters or repeat its function as long as the button is held down.

U

uninterruptible power supply. A buffer between utility power or other power source and a load that requires uninterrupted, precise power.

universal product code (UPC). An encoded number that can be assigned to and printed on or attached to an article of merchandise for scanning.

universal serial bus. An industry standard that makes it easy to expand PC functionality. The USB is a 12-Mbps serial bus designed to replace almost all low-to-medium speed connections to peripheral devices such as keyboards, mice, and printers.

unnumbered acknowledgment. A data link control (DLC) command used in establishing a link and in answering receipt of logical link control (LLC) frames.

unnumbered (U) frame. A frame in unnumbered format, used to transfer unnumbered control functions. See also *information frame*, *supervisory frame*.

unprotected field. A display field that the terminal operator can add to, modify, or erase using the keyboard. Contrast with *protected field*.

UPC. Universal product code.

UPS. Uninterruptible power supply.

USB. universal serial bus

user. (1) Category of identification defined for file access protection. (2) A person using a program or system.

user exit. A point in an Toshiba-supplied program at which a user-written program may be given control.

utility program. (1) A computer program in general support of the processes of a computer; for instance, a diagnostic program, a trace program, a sort program. (2) A program designed to perform an everyday task such as copying data from one storage device to another.

V

variable. (1) A named entity that is used to refer to data and to which values can be assigned. Its attributes remain constant, but it can refer to different values at different times. (2) In computer programming, a character or group of characters that refers to a value and, in the execution of a computer program, corresponds to an address. (3) A quantity that can assume any of a given set of values.

version. A separate Toshiba-licensed program, based on an existing Toshiba-licensed program, that usually has significant new code or new function.

VFD. Vacuum fluorescent display.

VFS. virtual file system.

video display. (1) An electronic transaction display that presents visual information to the point-of-sale terminal operator and to the customer. (2) An electronic display screen that presents visual information to the display operator.

virtual circuit. Synonym for *virtual connection*.

virtual connection. (1) A connection between two nodes on the network that is established using the transport layer and provides reliable data between nodes. (2) A logical connection established between two data terminal equipment (DTE) devices. Synonymous with *virtual circuit*.

virtual drive. Computer memory used as if it were a direct access storage device. Synonym for *RAM disk*.

virtual file system (VFS). Within 4690 OS V2, the virtual file system is used to provide support for long file names by creating two virtual drives that support file names greater than eight characters in length.

virtual machine (VM). A functional simulation of a computer and its associated devices. Each virtual machine is controlled by a suitable operating system, for example, a conversational monitor system. VM controls concurrent execution of multiple virtual machines on one host computer.

virtual storage (VS). (1) The storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations. (2) Addressable space that is apparent to the user as the processor storage space, from which the instructions and the data are mapped into the processor storage locations. Contrast with *real storage*.

VM. Virtual machine.

VS. Virtual storage.

W

wand. A commercially available device used to read information encoded on merchandise tickets, credit cards, and employee badges.

wanding. Passing the tip of the wand reader over information encoded on a merchandise ticket, credit card, or employee badge.

wideband. Synonym for *broadband*.

work file. A file that is both created and deleted in the same job.

workstation. (1) An I/O device that allows either transmission of data or the reception of data (or both) from a host system, as needed to perform a job: for example, a display station or printer. (2) A configuration of I/O equipment at which an operator works. (3) A terminal or microcomputer, usually one connected to a mainframe or network, at which a user can perform tasks.

X

XID. Exchange identification.

X.21. In data communication, a recommendation of the CCITT that defines the interface between data terminal equipment (DTE) and public data networks for digital leased and circuit switched synchronous services.

X.21 bis. In data communication, an interim specification of the CCITT that defines the connection of data terminal equipment (DTE) to an X.21 (public data) network using V-series interchange circuits such as those defined by CCITT V.24 and CCITT V.35.

X.25. A CCITT Recommendation that defines the physical level (physical layer), link level (data link layer), and packet level (network layer), of the OSI Reference Model. An X.25 network is an interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) operating in the packet mode, and connected to public data networks by dedicated circuits. X.25 networks use the connection-mode network service.

Index

Numerics

- 3270 emulation session identifiers 188
- 3720 emulation
 - 4693 ANPOS keyboard 206
 - accessing with a 4680 BASIC interface 189
 - ADXHS30F.DAT 203
 - ADXHS30F.DAT file, altering 203
 - ANPOS keyboard 206
 - API verb timeouts 189
 - API verbs 188, 190
 - ASCII-EBCDIC translation functions 204
 - compiling and linking to 202
 - designing applications for 187
 - function calls 188
 - keyboard and language combinations 203
 - keyboards, supported 205
 - keycode mapping 205
 - messages 206
 - NLS file structure and contents 204
 - presentation space 188
 - reconfiguring the NLS file 206
 - S3.CONNECT 190
 - S3.COPYFIELD (Copy-Field) 200
 - S3.COPYSTRING (Copy-String) 200
 - S3.DISCONNECT 191
 - S3.LOCCURSOR (Locate-Cursor) 195
 - S3.LOCFIELD (Locate-Field) 194
 - S3.LOCSTR (Locate-String) 193
 - S3.QUERYOGL (Query-OGL) 196
 - S3.SENDKEY (Send-Key) 197
 - S3.SENDSTRING (Send-String) 199
 - S3.SETCURSOR (Set-Cursor) 195
 - S3.WAIT 192
 - session identifiers 188
 - starting from the API 189
 - starting from the command line 189
 - starting the 188
 - subprograms 188
 - supported keyboards 205
 - terminal support 206
 - using the API 187
 - writing applications for 3270 API 201
- 4680 BASIC interface to access 3270 emulation 189
- 4680 BASIC statements for LU 0 applications
 - CLOSE statement 215
 - example of an interface 215
 - GETLONG statement 214
 - OPEN LINK SNA statement 212
 - OPEN SESSION SNA statement 212
 - programming considerations for CPI Communications, LU 6.2 226
 - PUTLONG statement 215
 - READ statement for LU 0/SNA 213
 - RESUME RETRY statement 215
 - using 212
 - WRITE statement 214
 - XCMGLE for LU 6.2/SNA 223
- 4690 extensions to SAA CPI communications 218

4690 subdirectory 91
4693 ANPOS keyboard with 3270 emulation 206

A

accessing 3270 emulation with a 4680 BASIC interface 189
actfilt command 58
action series used with HCP
 SDLC protocol
 with data following 292
 with no data 291
activating
 audible alarm for all store controllers 154
 audible alarm for one store controller 154
 command for audible alarm function, LAN system 151
 command for audible alarm function, non-LAN system 152
 remote audible alarm example using RCP 154
acttun command 65
ADCS START USER PROGRAM command 211
ADD KEYED RECORDS command 99
adjust tape tension command 158
ADX_CCLOSE_LS 271
ADX_CGET_STAT 273
ADX_COPEN_LINK 269
ADX_COPEN_SESS 270
ADX_CREAD_HOST 271
ADX_CSEND_REQ 276
ADX_CWRITE_HOST 272
ADXCS20L 136, 163
ADXCS30L 165
ADXCSH0L 136
ADXC SM0L 145, 170
ADXC SN0L 146, 170
ADXC SP0L 139, 170
ADXC SQ0L 140
ADXC SR0L 169
ADXC SS0L 150, 170
ADXHS30F.DAT file, altering 203
adxipsaf.386 58
adxipsat.386 65
adxipscf.386 58
adxipsct.386 66
adxipsgf.386 55
adxipsgt.386 62
adxipslf.386 62
adxipslt.386 68
adxipsmf.386 61
adxipsrf.386 61
adxipsrt.386 68
adxipsst.386 69
ADXSSHCF.DAT 83
ADXSSHCL.386 73
ADXSSHDF.DAT 81
ADXSSHD L.386 71
ADXSSHFL.386 77
ADXSSHK L.386 79
ADXSSHPL.386 76
ADXSSHXH.DAT 84
alert numbers for master controller event numbers 181
alert numbers for subordinate controller event numbers 182
alert numbers for terminal event numbers 182

- alerts
 - system 181
 - user 181
- altering the ADXHS30F.DAT file 203
- ANPOS keyboard with 3270 emulation 206
- API
 - See application program interface
- API verb timeouts 189
- APPC applications 217
- application directory information, RCMS 92
- application program interface
 - ADXVB3AI.J86 202
 - applications for 3270 201
 - compilation units 202
 - compiling 202
 - connect verb 190
 - copy-field verb 200
 - copy-string verb 200
 - disconnect verb 191
 - linking 202
 - locate-cursor verb 195
 - locate-field verb 194
 - locate-string verb 193
 - national language support 203
 - query-OGI verb 196
 - send-key verb 197
 - send-string verb 199
 - session identifiers 188
 - set-cursor verb 195
 - starting 3270 (store controller only) 189
 - starting 3270 emulation 188
 - using with 3270 emulation 187
 - verb timeouts with 3270 emulation 189
 - verbs 188
 - verbs with 3270 emulation 190
 - wait verb 192
 - writing applications 201
- application programs, writing for 3270 API 201
- application started by unsolicited BIND request 210
- apply software maintenance command 154
- applying software maintenance from a host site 163, 164, 178
- ASCII-EBCDIC translation functions 204
- ASCII/EBCDIC translation table for printer, HCP 115
- ASYNCR
 - ASYNCR status 273, 277
 - checking read/write functions 266
 - controlling buffer use 266
 - example of application in 4680 BASIC 268
 - monitoring modem interface signals 267
 - optional telephone number 269
 - reading data from the host 265
 - RS-422 lines 264
 - sending requests to the driver 276
 - using C-language and COBOL 269, 272
 - using link number returned 273
 - with a link number 272
 - writing data to the host 266
- ASYNCR programs, non-SNA 6
- audible alarm
 - activate a store controller 154
 - activate all store controllers 154

- audible alarm (*continued*)
 - activate command for LAN system 151
 - activate command for non-LAN system 152
 - commands 151
 - deactivate a store controller 154
 - deactivate all store controllers 154
 - deactivate command 154
 - examples for LAN system 152
 - examples for non-LAN system 153
 - HCP file name for 178
 - report audible alarm status command 154
 - report audible alarm status for a store controller 154
 - report audible alarm status for all store controllers 154
 - setting alarm duration 154
 - using Remote Command Processor (RCP) 154
- audible alarm commands 151

B

- backup
 - command using streaming tape drive utility 156
- BASIC language
 - programming considerations for CPI communications, LU 6.2 227
 - programming considerations for CPI Communications, LU 6.2 226
- BASIC programming examples 370
- battery return program 400
- BIND
 - HCP request unit format 283
 - processing for LU 0 209
 - request 213
 - unsolicited request starts LU 0 application 210
- bind buffer 270
- bind processing for LU 0 209
- BOOTP client with TCP/IP applications 20
- BOOTP server with TCP/IP applications 21
- BSX filenames (HCP six-character) 123

C

- C & SM
 - See Communications and Systems Management
- C & SM support for SNA link flags 275
- C language
 - close session 271
 - CPI communications file requester example program 361
 - CPI communications file server example program 366
 - designing applications 269
 - example TPs 359
 - header file definitions example for transaction programs 359
 - obtain status from link or session 273
 - open communications link 269
 - open session 270
 - programming considerations for CPI communications, LU 6.2 227
 - programming considerations for CPI Communications, LU 6.2 226
 - read data from link or session 271
 - requests to the driver 276
 - write data to link or session 272
- cable ferrite requirement 398
- calls 217
- characteristics of terminal, setting 163
- chfilt command 58

- chtun command 66
- Class A compliance statement
 - Australia and New Zealand 394
 - China 397
 - European Union 394, 398
 - FCC (USA) 394
 - Germany 395
 - Industry Canada 394
 - Japan 398
 - Russia 396
 - Taiwan 397
- close link or session 271
- close series used with HCP
 - SDLC protocol
 - close series 287
- CLOSE statement 215
- closing
 - SNA sessions and links 211
- closing LU 0 sessions and subarea links 211
- COBOL language
 - close session 271
 - CPI communications file requester example program 386
 - CPI communications file server example program 389
 - designing applications 269
 - examples 384
 - header file definitions for transaction programs example 384
 - obtain status from link or session 273
 - open communications link 269
 - open session 270
 - programming considerations for CPI communications, LU 6.2 227
 - programming considerations for CPI Communications, LU 6.2 226
 - read data from link or session 271
 - requests to the driver 276
 - write data to link or session 272
- coding tips for writing non-4680 BASIC programs 269
- command file for RCP 132
- command for streaming tape drive
 - example of adjust tape tension 159
 - example of adjust tape tension command 158
 - example of backup command 156, 160
 - example of erase command 158, 162
 - example of format 162
 - example of list command 157, 161
 - example of restore command 157, 161
- command mode file
 - special logical names for 118
- command to load all terminals 162
- commands
 - activate audible alarm for LAN system 151
 - activate audible alarm for LAN system, examples 152
 - activate audible alarm for non-LAN system 152
 - activate audible alarm for non-LAN system, examples 153
 - ADCS START USER PROGRAM for starting RCP 172
 - ADD KEYED RECORDS (HCP) 99
 - adjust tape tension (streaming tape drive utility) 158
 - apply software maintenance 154
 - audible alarm 151
 - backup (streaming tape drive utility) 156
 - BIND request for starting RCP 172
 - configuration activation for RCP 149
 - configuration data formatter for RCP 149

commands (*continued*)

- CREATE FILE (HCP) 100
- deactivate audible alarm 154
- DELETE 92
- DELETE KEYED RECORDS (HCP) 102
- DSX commands used with RCMS 90
- DUMP FILE (HCP) 102
- dump formatter 138
- dump formatter, remote (RCP) 138
- erase 161
- erase (streaming tape drive utility) 158
- ethernet trace 142
- file compression/decompression, RCP 165
- file management command for trace for RCP 144, 145
- file management for system log for RCP 147
- format (optical backup utility) 162
- format performance data 139
- HCP 99
- HCP EXTENDED DUMP 105
- INFORM OPERATOR 93
- INITIATE FUNCTION 93
- INITIATE FUNCTION (asynchronous) 93
- INITIATE FUNCTION (synchronous) 93
- INITIATE FUNCTION for starting RCP 172
- list 161
- list (streaming tape drive utility) 157
- list file format for file compression/decompression, RCP 167
- load all terminals 162
- LOAD FILE (HCP) 106
- optical drive utility 159
- processing multiples for RCP 177
- PURGE FILE (HCP) 107
- QUERY 94
- RCMS 90
- re-IPL command for RCP 136
- READ KEYED RECORD (HCP) 107
- Remote Change Management Server 90
- Remote Command Processor 136
- remote STC 137
- REPLACE KEYED RECORD (HCP) 108
- report audible alarm status 154
- report module level for RCP 150
- restore (optical backup utility) 160
- restore (streaming tape drive utility) 157
- RETRIEVE 91
- run remote STC in one terminal, remote (RCP) 137
- SEND for existing file 90
- SEND for new file 90
- start performance for RCP 138
- start trace for RCP: all traces 141
- start trace for RCP: device channel 141
- start trace for RCP: loop, hard disk drive, host line 140
- START USER PROGRAM (SUP) 211
- start/stop trace for RCP 138
- STATUS (HCP) 108
- stop performance command for RCP 143
- stop trace for RCP 143
- streaming tape drive utility 156
- system log formatter for RCP 146
- system log formatter for RCP, example 148
- trace for RCP 138

- commands (*continued*)
 - trace formatter for RCP 144
- communications
 - CPI calls for LU 6.2 217
 - host communications using SDLC/SNA communications protocols 287
 - introduction to 3
 - link and session considerations 5
 - LU 0 209
 - LU 6.2 217
 - management 181
- Communications and Systems Management
 - alert numbers for master controller event numbers 181
 - alert numbers for subordinate controller event numbers 182
 - alert numbers for terminal event numbers 182
 - MAC address data file 184
 - system alerts 181
 - user application alerts 181
 - vital product data file 183
- compiling and linking in 3270 emulation 202
- compressing files
 - file compression/decompression, RCP 165
 - list file format for file compression/decompression, RCP 167
 - on diskette, RCP 167
 - on hard disk drive, RCP 167
 - remotely, RCP 165
- concurrent conversations 226
- conditional disable link routine for LU 6.2 219
- configuration
 - HCP file name for 178
 - set date and time 164
 - set terminal characteristics 163
- configuration activation command, RCP 149
- configuration data formatter command, RCP 149
- configuration, PXE 23
- configuring DHCP server 26
- configuring TCP/IP for 4690 OS controllers 10
- configuring wireless terminals for PXE 26
- configuring, DHCP server 23
- connect, API verb 190
- controller and auxiliary console support 205
- conversations, concurrent 226
- copy-field, API verb 200
- copy-string, API verb 200
- CPI communications calls for LU 6.2 217
- CPI-communications
 - 4690 extensions 218
 - description of calls for LU 6.2 217
 - example of file requester program 361
 - example of file server program 366
 - programming considerations for BASIC, LU 6.2 227
 - programming considerations for C language, LU 6.2 227
 - programming considerations for COBOL, LU 6.2 227
 - programming considerations for languages 226
 - pseudonyms for calls 217
 - subroutine definitions for 4680 BASIC 372
- CREATE FILE command 100

D

- damage from electrostatic discharge 398

- data
 - area (HCP) 125
 - binary format and RCMS 95
 - EBCDIC format and RCMS 95
 - format file conversion 95
 - PSS type and RCMS 95
- data encryption support for LU 0 209
- data flow control support for LU 0 209
- data terminal ready 273
- deactivate
 - audible alarm for one or all store controllers 154
 - command for audible alarm function 154
 - remote audible alarm using RCP command file 154
- decompressing files
 - file compression/decompression, RCP 165
 - list file format for file compression/decompression, RCP 167
 - remotely, RCP 165
- deleting
 - DELETE command (DSX) 92
 - DELETE KEYED RECORDS command 102
- designing an asynchronous program 263
- designing applications with C and COBOL 269
- designing LU 0/SNA programs 209
- DHCP configuration file 24
- DHCP configuration options 25
- DHCP server 22
 - backup configuration 25
 - configuration file 24
 - configuration options 25
 - configuring 23
 - configuring for terminal IP address assignment 26
 - PXE booting 24
 - troubleshooting 29
- DHCP server backup configuration 25
- DHCP setup for PXE booting 24
- DHCPINFO utility 27
- direction indicator 124
- directory information, RCMS
 - for application directories 92
 - for root directories 91
 - for system directories 91
 - for user directories 92
- disable link routine for LU 6.2 219
- disconnect, API verb 191
- disposal of equipment 399
- disposition of streaming tape drive data
 - add new volume 157
 - back up to new volume 159
 - erase 157
 - format 159
 - open files 159
 - overwrite 157
 - unlocked 159
 - volume 1 159
 - write access 159
- Distributed Systems Executive (DSX)
 - commands used with RCMS 90
 - interface on operating system 89
 - SEND command for existing file with RCMS 90
 - SEND command for new file with RCMS 90
 - used with RCMS 89

- dump
 - DUMP FILE command 102
 - filenames (HCP six-character) 123
 - formatter command used with RCP 138
 - HCP EXTENDED DUMP command 105
 - remote store controller using RCP 138
 - remote terminal dump using RCP 138
 - summary for HCP filename 178
- dump series used with HCP
 - SDLC protocol
 - dump series 289

E

- electromagnetic Interference statement
 - Russia 396
- electronic emissions notices 394
 - Australia and New Zealand 394
 - China 397
 - European Union 394, 398
 - FCC (USA) 394
 - Germany 395
 - Industry Canada 394
 - Japan 398
 - Korea 396
 - Taiwan 397
- electrostatic discharge (ESD) 398
- elooaddr 11
- emulation, 3270
 - 4693 ANPOS keyboard 206
 - accessing with a 4680 BASIC interface 189
 - ADXHS30F.DAT file, altering 203
 - ANPOS keyboard 206
 - API verb timeouts 189
 - API verbs 188, 190
 - ASCII-EBCDIC translation functions 204
 - compiling and linking to 202
 - designing applications for 187
 - function calls 188
 - keyboard and language combinations 203
 - keycode mapping 205
 - messages 206
 - NLS file structure and contents 204
 - presentation space 188
 - reconfiguring the NLS file 206
 - S3.CONNECT 190
 - S3.COPYFIELD (Copy-Field) 200
 - S3.COPYSTRING (Copy-String) 200
 - S3.DISCONNECT 191
 - S3.LOCCURSOR (Locate-Cursor) 195
 - S3.LOCFIELD (Locate-Field) 194
 - S3.LOCSTR (Locate-String) 193
 - S3.QUERYOGL (Query-OGL) 196
 - S3.SENDKEY (Send-Key) 197
 - S3.SENDSTRING (Send-String) 199
 - S3.SETCURSOR (Set-Cursor) 195
 - S3.WAIT 192
 - session identifiers 188
 - starting from the API 189
 - starting from the command line 189
 - starting the 188

- emulation, 3270 (*continued*)
 - subprograms 188
 - terminal support 206
 - using the API 187
 - writing applications for 3270 API 201
- enable link routine for LU 6.2 220
- end of file (HCP) 126
- end of group (EOG) indicator 124
- end of life disposal 399
- end of record (EOR) indicator 124
- ending
 - end of file (EOF) indicator 124
 - ending links from LU 6.2 applications 226
- enhanced loopback address
 - ifconfig 12
- equipment disposal 399
- erasing
 - erase command using optical backup utility 161
 - using streaming tape drive utility 158
- error handling (HCP) 128
- establishing an SNA link from LU 6.2 applications 226
- Ethernet SNA tunable parameters 4
- ethernet trace, RCP 142
- European Union battery recycling statement 400
- examples
 - 4680 BASIC pseudonym equates 375
 - activate audible alarm (remote) command 154
 - activate audible alarm for LAN system 152
 - activate audible alarm for non-LAN system 153
 - adjust tape tension command (Streaming Tape Drive) 158, 159
 - ASYNCR application in 4680 BASIC 268
 - backup command (optical diskette drive) 160
 - backup command (Streaming Tape Drive) 156
 - compress all files on diskette drive, RCP 167
 - compress all files on hard disk drive, RCP 167
 - configuration activation command 149
 - configuration data formatter command 150
 - CPI communications file requester program 361
 - CPI communications file requester program, COBOL 386
 - CPI communications file server program 366
 - CPI communications file server program, COBOL 389
 - CPI communications subroutine definitions for 4680 BASIC 372
 - deactivate audible alarm (remote) command 154
 - dump formatter command for RCP 138
 - erase command (optical diskette drive) 162
 - erase command (Streaming Tape Drive) 158
 - ethernet trace 142
 - ethernet trace SAP 143
 - file requester program in 4680 BASIC 376
 - file server program in 4680 BASIC 380
 - format command (optical diskette drive) 162
 - header file definitions for TPs in C language 359
 - header file definitions for transaction program, COBOL 384
 - list command (optical diskette drive) 161
 - list command (Streaming Tape Drive) 157
 - load all terminals command 163
 - module level formatter command 151
 - RCP application flow 131
 - remote trace formatting for RCP 144
 - report audible alarm (remote) status command 154
 - restore command (optical diskette drive) 161

examples (*continued*)

- restore command (streaming tape drive) 157
- SNA interface with the host 215
- system log command 147
- system log formatter, RCP 148
- variable type for 4680 BASIC 370

F

- ferrite requirement 398
- FID type 2 headers 209
- file compression/decompression command 165
- file data received from the HCP 97
- file management command for system log, RCP 147
- file management command for trace output data files for RCP 144, 145
- file names
 - for HCP 109
 - HCP file names used with RCP files 178
 - HCP six-character dump and BSX 123
 - listing using RETRIEVE command, RCMS 91
 - point-of-sale application file names for HCP 111
 - system file names for HCP 110
 - user file names for HCP 113
- file security for HCP 118
- file structure and contents, NLS 204
- file use table for HCP 114
- files
 - host access to store controller files, RCMS 95
 - naming for HCP 109
 - point-of-sale application file names for HCP 111
 - recovery and RCMS 94
 - Remote Change Management Server and 94
 - system file names for HCP 110
 - transmitting from host on LAN system 100
 - user file names for HCP 113
- filters
 - configuring 49
 - description 44
 - intrusion detection and prevention 52
 - rules
 - auto-generated 49
 - examples 50
 - pattern matching 52
 - predefined 50
 - shun host 53
 - shun port 53
 - stateful 55
 - subnet masks 50
 - timed 55
 - user-defined 49
- first message (HCP) 127
- flat panel displays 401
- format performance data 139
- format types (HCP) 125
- formatting
 - a trace from a remote site using RCP 144
 - dumps from a host site 138
 - system log formatter command, example 148
 - system log formatter for RCP 146
- FTP client with TCP/IP applications 15
- FTP server with TCP/IP applications 16

- full store controller dump using RCP 138
- full terminal dump using RCP 138
- function calls 188
- functions of RCP 131

G

- genfilt command 55
- gentun command 62
- get last error routine for LU 6.2 220
- GETLONG statement 214
- getting
 - status from link or session 273
- guidelines for writing non-4680 BASIC programs 269

H

- HCP
 - See Host Command Processor
- HCP EXTENDED DUMP command 105
- host
 - access to store controller files, RCMS 95
 - applying software maintenance from 178
 - communications to store controller 279
 - communications using SDLC/SNA communications protocols 287
 - defining store controller 131
 - LU 0/SNA programs 209
- Host Command Processor
 - ADCS START USER PROGRAM command 99, 211
 - ADD KEYED RECORDS command 99
 - ASCII/EBCDIC translation table for printer 115
 - BIND request unit format 283
 - commands 99
 - CREATE FILE command 100
 - data area 125
 - DELETE KEYED RECORDS command 102
 - direction indicator 124
 - DUMP FILE command 102
 - end of file 126
 - end of file (EOF) indicator 124
 - end of record (EOR) indicator 124
 - error handling 128
 - file data received from 97
 - file names for RCP files 178
 - file names that do not follow ADX naming conventions 123
 - file security 118
 - file use table 114
 - first message 127
 - format types 125
 - HCP EXTENDED DUMP command 105
 - inbound pipe 124
 - indicator byte 125
 - introduction 97
 - key only indicator 125
 - LOAD FILE command 106
 - logical unit characteristics 98
 - message format 124
 - message length high-order byte 125
 - message length low-order byte 125
 - naming files 109
 - outbound pipe 124

Host Command Processor *(continued)*

- path information units, LU address 279
- pipe interface 123
- point-of-sale file names 111
- PURGE FILE command 107
- READ KEYED RECORD command 107
- record header 124
- record level command (RLC) indicator 124
- REPLACE KEYED RECORD command 108
- sample flows 129
- SDLC/SNA communications protocols 287
- SNA TH/RH/RU contents 279
- special logical names 118
- STATUS command 108
- subdirectory listing 109
- system file names 110
- translation application message format 124
- translation interface 123
- translation table 114
- transmitting PC-format sequential files 97
- transmitting PSS-format sequential files 98
- user file names 113
- using SNA request/response units 279
- using the HCP 97

I

- identifying a network router, TCP/IP 13
- inbound pipe
 - to HCP 124
 - to RCMS 95
- indicator byte 125
- INETD superserver with TCP/IP applications 15
- INFORM OPERATOR command (DSX) 93
- information messages for 3270 emulation 206
- INITIATE FUNCTION command 93
 - NetView DM only, asynchronous 93
 - NetView DM only, synchronous 93
- interface
 - HCP translation 123
 - pipe 123
 - RCMS translation 95
- Internet Protocol Security (IPsec)
 - actfilt command 58
 - actun command 65
 - adxipsaf.386 58
 - adxipsat.386 65
 - adxipscf.386 58
 - adxipsct.386 66
 - adxipsgf.386 55
 - adxipsgt.386 62
 - adxipslf.386 62
 - adxipslt.386 68
 - adxipsmf.386 61
 - adxipsrf.386 61
 - adxipsrt.386 68
 - adxipsst.386 69
 - chfilt command 58
 - chun command 66
 - commands 55
 - configuring filters 49

- Internet Protocol Security (IPsec) *(continued)*
 - configuring manual tunnels 46
 - enabling 44
 - features 44
 - genfilt command 55
 - gentun command 62
 - ipsecstat command 69
 - limitations 44
 - lsfilt command 62
 - lstun command 68
 - mvfilt command 61
 - overview 43
 - rmfilt command 61
 - rmtun command 68
- introduction to communications 3
- introduction to the Host Command Processor 97
- IPL processing for RCP 176
- IPsec
 - actfilt command 58
 - acttun command 65
 - adxipsaf.386 58
 - adxipsat.386 65
 - adxipscf.386 58
 - adxipsct.386 66
 - adxipsgf.386 55
 - adxipsgt.386 62
 - adxipslf.386 62
 - adxipslt.386 68
 - adxipsmf.386 61
 - adxipsrf.386 61
 - adxipsrt.386 68
 - adxipsst.386 69
 - chfilt command 58
 - chtun command 66
 - commands 55
 - configuring filters 49
 - configuring manual tunnels 46
 - enabling 44
 - features 44
 - genfilt command 55
 - gentun command 62
 - ipsecstat command 69
 - limitations 44
 - lsfilt command 62
 - lstun command 68
 - mvfilt command 61
 - overview 43
 - rmfilt command 61
 - rmtun command 68
- ipsecstat command 69

J

- Japan Electronics and Information Technology Industries Association statement 396
- Japanese Electrical Appliance and Material Safety Law statement 398
- Japanese power line harmonics compliance statement 398
- Japanese VCCI Council Class A statement 396

K

- key only (RLC indicator) 125
- keyboard and language combinations in 3270 emulation 203
- keyboards supported for 3270 emulation 205
- keycode mapping 205
- Korean communications statement 396

L

- lan system
 - bit format for CREATE FILE command 100
 - retrieving file attributes 99
- language and keyboard combinations in 3270 emulation 203
- lines
 - RS-422 (ASYNC) 264
- link and session considerations for communication 5
- link considerations for LU 0/SNA programs 209
- links
 - closing an LU 0 subarea link 211
 - ending from LU 6.2 applications 226
 - establishing an SNA link from LU 6.2 applications 226
 - opening an SNA subarea link for LU 0 211
 - status 273
- list file format for file compression/decompression, RCP 167
- listing
 - file format for file compression/decompression 167
 - filenames using RETRIEVE command, RCMS 91
 - list command using optical backup utility 161
 - list command using streaming tape drive utility 157
- load series used with HCP
 - SDLC protocol
 - complete with no detected errors 288
 - complete with store controller detected error 289
- loading
 - all terminals remotely 162
 - example of load command in RCP command file 163
 - load all terminals command 162
 - LOAD FILE command 106
- locate-cursor, API verb 195
- locate-field, API verb 194
- locate-string, API verb 193
- logical names, special 118
- logical unit
 - characteristics of for HCP 98
 - path information units, address for HCP 279
 - Remote Change Management Server 89
- logical unit characteristics of the HCP 98
- lsfilt command 62
- lstun command 68
- LU
 - See logical unit
- LU 0/SNA communications
 - 4680 BASIC statements 212
 - bind processing 209
 - CLOSE statement 215
 - closing LU 0 sessions 211
 - closing LU 0 subarea links 211
 - data flow control support 209
 - designing 209
 - example of a SNA interface 215

- LU 0/SNA communications *(continued)*
 - GETLONG statement 214
 - line communications protocol 209
 - link and session considerations 209
 - OPEN LINK SNA statement 212
 - OPEN SESSION SNA statement 212
 - opening an LU 0 session 211
 - opening an SNA subarea link 211
 - PUTLONG statement 215
 - READ statement 213
 - RESUME RETRY statement 215
 - SNA support 209
 - START USER PROGRAM command 211
 - starting applications 210
 - transmission control support 209
 - unsolicited BIND request 210
 - using BASIC statements with 212
 - WRITE statement 214
- LU 0/SNA program 209
- LU 6.2 communications
 - 4690 extensions 218
 - APPC applications 217
 - BASIC programming considerations 227
 - C language programming considerations 227
 - COBOL programming considerations 227
 - conditional disable link (XCMCDL) 219
 - CPI Communications calls 217
 - disable link (XCMDL) 219
 - enable link (XCMELE) 220
 - ending links 226
 - establishing an SNA link 226
 - get last error (XCMGLE) 220
 - library routines 217
 - programming considerations for CPI Communications 226
 - query link status (XCMQL) 224
 - set time out value (XCMSTO) 224
 - starting a transaction program 225
 - using 217
 - XCMCDL 219
 - XCMDL 219
 - XCMELE 220
 - XCMGLE 220
 - XCMGLE in 4680 BASIC 223
 - XCMQL 224
 - XCMSTO 224

M

- MAC address data file 184
- MAC address data record format 184
- management
 - communications 181
 - systems 181
- mercury-added statement 401
- message format for HCP 124
- message length high-order byte 125
- message length low-order byte 125
- message, first (HCP) 127
- messages for 3270 emulation 206
- modem interface signals for ASYNC 267
- module level, HCP file name for 178

- MTFTP server 31
 - PXE booting 31
 - startup and configuration 31
- multiple commands and command files for RCP 177
- mvfilt command 61

N

- naming HCP files 109
- national language support
 - 3270 emulation 203
 - 3270 emulation, keyboard and language combinations 203
 - ADHS30F.DAT, reconfiguring 206
 - ADXHS30F.DAT 203
 - altering the ADXHS30F.DAT file 203
 - ASCII-EBCDIC conversions 204
 - contents 204
 - controller and auxiliary console support 205
 - file structure and contents 204
 - keyboard and language combinations 203
 - keycode mapping 205
 - supported keyboards 205
- NetView Distribution Manager
 - commands used with RCMS 90
 - interface on operating system 89
- NetView DM
 - See NetView Distribution Manager
- NFS client with TCP/IP applications 19
- NFS server with TCP/IP applications 18
- NLS
 - See national language support
- NLS file, how to reconfigure 206
- non-4680 BASIC programs 269
- non-SNA communications protocol
 - ASYNCR programs 6
 - X.25 programs 6
- notices 393
 - battery recycling 400
 - cable ferrites 398
 - electronic emissions 394
 - electrostatic discharge (ESD) 398
 - end of life disposal 399
 - Toshiba 393

O

- open communications link 269
- open link 269
- OPEN LINK statement 212
- open series used with HCP
 - SDLC protocol
 - open series 287
- open session 270
- OPEN SESSION statement 212
- opening an LU 0 session 211
- opening an SNA subarea link 211
- optical drive utility commands
 - commands 159
 - erase command 161
 - format command 162
 - list command 161

- optional telephone number 269
- outbound pipe to HCP 124
- outbound pipe to RCMS 95

P

- parameter for Remote Command Processor 136
- parameters, SNA, Ethernet, tunable 4
- path information units, HCP LU address 279
- PC-format sequential files 97
- perchlorate 401
- performance, PXE 23
- pingTime parameter, PXE 23
- pipe
 - inbound to HCP 124
 - inbound to RCMS 95
 - interface 123
 - outbound to HCP 124
 - outbound to RCMS 95
- point-of-sale file names for HCP 111
- presentation space 188
- profiles for request/response header 279
- profiles for transmission header 279
- programming
 - considerations for BASIC, CPI communications 227
 - considerations for C language, CPI communications 227
 - considerations for C language, CPI Communications 226
 - considerations for COBOL, CPI communications 227
- programming considerations for CPI Communications, LU 6.2 226
- programs
 - designing LU 0/SNA 209
 - designing LU 6.2/SNA 217
 - examples in BASIC 370
 - variable type for 4680 BASIC 370
 - writing non-4680 BASIC 269
- protocol, SNA for LU 0 209
- protocols
 - HCP 287
 - HCP SDLC/SNA 287
 - SNA for LU 0 209
- PSS-format sequential files 98
- PURGE FILE command 107
- PUTLONG statement 215
- PXE
 - configuration 23
 - configuring wireless terminals 26
 - DHCP setup 24
 - IP address 23
 - performance considerations 23
 - pingTime parameter 23
 - restrictions 22
- PXE booting 24
- PXE environment 22
- PXE environment restrictions 22

Q

- QUERY command (NetView DM) 94
- query link status routine for LU 6.2 224
- query-OGI, API verb 196

R

RCMS

See Remote Change Management Server

RCMS commands

DELETE 92

INFORM OPERATOR 93

INITIATE FUNCTION 93

INITIATE FUNCTION (asynchronous) 93

INITIATE FUNCTION (synchronous) 93

QUERY 94

RETRIEVE 91

SEND for existing file 90

SEND for new file 90

RCMS translation interface 95

RCP

See Remote Command Processor

re-ipl command for RCP 136

read data from link or session 271

READ KEYED RECORD command 107

READ statement 213

reconfiguring the NLS file 206

record header for HCP 124

record level command (RLC) indicator 124

Remote Change Management Server 89

binary data format 95

commands 90

data format file conversion 95

EBCDIC data format 95

file recovery 94

files and 94

functions of 89

host access to store controller files 95

logical unit characteristics 89

PSS data type 95

requesting a session with 89

translation interface 95

Remote Command Processor

activate audible alarm 154

apply software maintenance command 154

applying software maintenance using 163, 164, 178

audible alarm commands 151

command file 132

commands and parameters 136

compress/decompress files remotely 165

configuration activation command 149

configuration data formatter command 149

deactivate audible alarm 154

dump formatter command 138

ethernet trace command 142

example of configuration activation command 149

example of configuration data formatter command 150

example of ethernet trace 142

example of ethernet trace command 143

example of flow 131

example of system log command 147

example of system log formatter command 148

file compression/decompression command 165

file management command for system log 147

file management command for trace output data files 144, 145

format performance data 139

functions of 131

Remote Command Processor *(continued)*

- HCP file name for 178
- HCP file names 178
- how to use 131
- IPL processing 176
- list file format for file compression/decompression command 167
- load all terminals remotely 162
- multiple commands and command files 177
- optical drive utility commands 159
- RCP selection file used with 131
- re-IPL command 136
- report audible alarm status 154
- report module level command 150
- retrieving reports formatted by 177
- run remote STC in one terminal command 137
- sample of 132
- set date and time 164
- set terminal characteristics 163
- start performance command 138
- start trace command 138
- start trace command for all traces 141
- start trace command for device channel 141
- start trace command for loop, hard disk drive, host line 140
- start/stop trace command 138
- starting 172
- starting from a host processor 172
- starting from an application 172
- status file used with RCP 134
- stop performance command 143
- stop trace command 138, 143
- streaming tape drive utility commands 156
- system log formatter 146
- trace commands 138
- trace formatter command 144
- user created output files 133
- using command files on LAN systems 133
- remote configuration activation 149
- remote dump formatting
 - store controller 138
 - terminal dump using RCP 138
- remote ethernet trace command 142
- remote reporting of configuration data 149
- remote STC using RCP 137
- remote terminal invocation 163
- REPLACE KEYED RECORD command 108
- report audible alarm (remote) status
 - example of in RCP command file 154
- report module level
 - example of in RCP command file 151
- report module level command used with RCP 150
- report status command for audible alarm function 154
- report status of audible alarm
 - for a store controller 154
 - for all store controllers 154
- report store controller audible alarm status 154
- reports formatted by RCP 177
- request, BIND 213
- request/response header profiles 279
- request/response unit
 - HCP BIND RU format 283
 - HCP/UP SNA 279

- request/response unit *(continued)*
 - profiles 279
- requests to the driver 276
- restore command using optical backup utility 160
- restore command using streaming tape drive utility 157
- restrictions for writing non-4680 BASIC programs 269
- RESUME RETRY statement 215
- RETRIEVE
 - command with RCMS 91
 - getting directory of filenames, RCMS 91
- retrieving
 - attributes on a LAN system 99
 - RCP reports 177
- rmfilt command 61
- rmtun command 68
- root directory information, RCMS 91
- RS-422 lines 264

S

- S3.CONNECT 190
- S3.COPYFIELD 200
- S3.COPYSTRING 200
- S3.DISCONNECT 191
- S3.LOCCURSOR 195
- S3.LOCFIELD 194
- S3.LOCSTR 193
- S3.QUERYOGL 196
- S3.SENDKEY 197
- S3.SENDSTRING 199
- S3.SETCURSOR 195
- S3.WAIT 192
- safety information xv
- sample flows (HCP) 129
- sample of RCP command file 132
- SDLC/SNA communications protocols used with HCP 287
- Secure FTP
 - ADXSSHFL.386 77
 - authorizing users 85
 - console session 77
 - running the SFTP console session as a background application 85
 - secure FTP server 76
- Secure FTP server 76
- Secure Shell (SSH)
 - ADXSSHCF.DAT 83
 - ADXSSHCL.386 73
 - ADXSSHDF.DAT 81
 - ADXSSHD.386 71
 - ADXSSHFL.386 77
 - ADXSSHKL.386 79
 - ADXSSHPL.386 76
 - ADXSSHXH.DAT 84
 - authorizing users 85
 - client 73, 84
 - client configuration file 83
 - introduction 71
 - Key generator 79
 - Secure FTP console session 77
 - Secure FTP server 76
 - server 71
 - server configuration file 81

- selection file used with RCP 131
- SEND command (DSX) for existing file 90
- SEND command (DSX) for new file 90
- send requests to the driver 276
- send-key, API verb 197
- send-string 199
- sequential files
 - transmitting PC-format 97
 - transmitting PSS-format 98
- session
 - closing an LU 0 session 211
 - identifiers 188
 - open, C and COBOL language programs 270
 - opening an LU 0 session 211
 - Remote Change Management Server 89
 - status, C and COBOL 273
- session considerations for LU 0/SNA programs 209
- session identifiers for 3270 emulation 188
- set system time and date 164
- set terminal characteristics 163
- set time out value routine 224
- set-cursor, API verb 195
- setting store controller audible alarms 154
- SFTP
 - ADXSSHFL.386 77
 - authorizing users 85
 - console session 77
 - running the SFTP console session as a background application 85
 - Secure FTP server 76
- SNA 269, 272, 273, 276
- SNA applications and BASIC statements 212
- SNA communications protocol
 - action series 290
 - action series with data following 292
 - action series with no data 291
 - close series 287
 - dump series 289
 - load series 287
 - load series complete with no detected errors 288
 - load series complete with store controller detected errors 289
 - open series used with HCP 287
- X.25 5
- SNA link flags 275
- SNA protocols, X.25 5
- SNA status 274, 277
- SNA support for LU 0 209
- SNA tunable parameters, Ethernet 4
- special logical names 118
- SSH
 - ADXSSHCF.DAT 83
 - ADXSSHCL.386 73
 - ADXSSHDF.DAT 81
 - ADXSSHD.386 71
 - ADXSSHFL.386 77
 - ADXSSHKL.386 79
 - ADXSSHPL.386 76
 - ADXSSHXH.DAT 84
 - authorizing users 85
 - client 73, 84
 - client configuration file 83
 - introduction 71

SSH (*continued*)

- key generator 79
- Secure FTP console session 77
- Secure FTP server 76
- server 71
- server configuration file 81
- start performance command for RCP 138
- start trace command for RCP: all traces 141
- start trace command for RCP: device channel 141
- start trace command for RCP: loop, hard disk drive, host line 140
- START USER PROGRAM command 99
- START USER PROGRAM command (SUP) 211
- start/stop trace command for RCP 138
- starting 3270 emulation 188
- starting 3270 emulation from the API 189
- starting 3270 emulation from the command line 189
- starting an LU 6.2 transaction program 225
- starting LU 0 applications 210
- starting RCP 172
- starting RCP from a host processor 172
- starting RCP from an application 172
- status 273
- STATUS command 108
- status file used with RCP 134
- status from link or session 273
- stop performance command for RCP 143
- stop trace command for RCP 143
- store controller
 - activate audible alarm 154
 - host access to files, RCMS 95
 - remote dump using RCP 138
 - report audible alarm status 154
- streaming tape drive utility
 - adjust tape tension command 158
 - backup command 156
 - commands 156
 - erase command 158
 - list command 157
 - restore command 157
- subdirectory listing of filenames for HCP 109
- subdirectory, 4690 91
- subnet masks 50
- subprograms 188
- SUP (START USER PROGRAM) command 211
- supported keyboards for 3270 emulation 205
- system alerts 181
- system directory information, RCMS 91
- system file names for HCP 110
- system log command, example in RCP command file 147
- system log from a remote site for RCP 146
- system log, example of formatter command 148
- system log, file management command 147
- system log, HCP file name for 178
- systems management 181

T

- Taiwanese battery recycling statement 400
- tape drive, adjust tape tension command 158
- tape drive, backup command 156
- tape drive, erase command 158

- tape drive, list command 157
- tape drive, restore command 157
- tape status, HCP file name for 178
- TCP/IP
 - configuring 10
 - identifying a network router 13
 - programming libraries 40
 - setting up other TCP/IP applications 14
 - using asynchronous communication 12
 - using host names 13
 - using in the operating system 7
 - using other TCP/IP applications 14
- Telnet client with TCP/IP applications 20
- terminal characteristics, setting 163
- terminal support for 3270 emulation 206
- terminal, remote dump 138
- terminal, remote STC 137
- TFTP server 31
 - PXE booting 31
 - startup and configuration 31
- time and date, setting 164
- timeouts, verb 189
- TP
 - examples of 359
 - on the 4690 store controller 217
 - programming considerations for 226
 - starting for LU 6.2 225
- trace commands for RCP 138
- trace formatter (remote)
 - example for RCP 144
 - file management command 144, 145
 - remote trace formatting 144, 145
- trace formatter command for RCP 144
- trace report, HCP file name for 178
- trademarks 402
- transaction programs
 - examples of 359
 - on the 4690 store controller 217
 - programming considerations for 226
 - starting for LU 6.2 225
- translation
 - ASCII-EBCDIC 204
 - functions, ASCII-EBCDIC 204
 - interface (HCP) 123
 - interface (RCMS) 95
- translation table for HCP files 114
- transmission control support for LU 0 209
- transmission header (TH) profiles 279
- transmitting files from the host on LAN system 100
- transmitting PC-format sequential files 97
- transmitting PSS-format sequential files 98
- troubleshooting DHCP server 29
- tunnels
 - configuring 46
 - description 44
 - host-firewall-host configuration 49
 - security associations 45, 46

U

- unit, HCP/UP SNA request/response 279
- user application alerts 181
- user created output files for RCP 133
- user directory information, RCMS 92
- user file names for HCP 113
- using host names, TCP/IP 13
 - configuring TCP keepalive timer value 14
 - defining search order 13
 - hosts file 13
 - resolv file 13
- using other TCP/IP applications 14
 - BOOTP client 20
 - BOOTP server 21
 - Enhanced Telnet server 36
 - FTP client 15
 - FTP server 16
 - INETD superserver 15
 - LPR client 38
 - MTFTP server 31
 - NFS client 19
 - NFS server 18
 - PCNFSD authentication server 39
 - rexec client 39
 - SNMP agent 33
 - Telnet client 20
 - Telnet server 35
 - TFTP server 31
- using RCP command files on LAN systems 133
- using the Remote Command Processor 131
- utility, DHCPINFO 27

V

- verb timeouts, API 189
- verbs
 - API 190
 - connect 190
 - copy-field 200
 - copy-string 200
 - disconnect 191
 - locate-cursor 195
 - locate-field 194
 - locate-string 193
 - query-OGI 196
 - S3.CONNECT 190
 - S3.COPYFIELD 200
 - S3.COPYSTRING 200
 - S3.DISCONNECT 191
 - S3.LOCCURSOR 195
 - S3.LOCFIELD 194
 - S3.LOCSTR 193
 - S3.QUERYOGL 196
 - S3.SENDKEY 197
 - S3.SENDSTRING 199
 - S3.SETCURSOR 195
 - S3.WAIT 192
 - send-key 197
 - send-string 199
 - set-cursor 195

verbs (*continued*)

used with API 188

wait 192

vital product data file 183

vital product data record format 183

W

wait, API verb 192

wireless terminals, configuring for PXE 26

write data to link or session 272

WRITE FORM statement 214

WRITE# statement 214

writing application programs for 3270 API 201

writing applications for 3270 API 201

writing non-4680 BASIC programs 269

X

X.25 5

XCMCDL 219

XCMDL 219

XCMEI 220

XCMGLE 220

XCMGLE in 4680 BASIC 223

XCMQL 224

XCMSTO 224

XOFF 273

XON 273



Product Number: 5639-P70

G362-0575-02

