



MacAPPC™  
Programmer's Reference  
and User's Guide



🍏 APPLE COMPUTER, INC.

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or in part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose.

© Apple Computer, Inc., 1989  
20525 Mariani Avenue  
Cupertino, CA 95014  
(408) 996-1010

Apple, the Apple logo, AppleLink, AppleTalk, HyperCard, ImageWriter, LaserWriter, and Macintosh are registered trademarks of Apple Computer, Inc.

APDA, EtherTalk, HyperCard, APPC, HyperTalk, MacAPPC, Macintosh Coprocessor Platform, MCP operating system, MPW, and Stackware are trademarks of Apple Computer, Inc.

ITC Avant Garde Gothic, ITC Garamond, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation.

Ethernet is a registered trademark of Xerox Corporation.

IBM is a registered trademark, and Token Ring is a trademark, of International Business Machines Corporation.

MicroVAX is a trademark of Digital Equipment Corporation.

NuBus is a trademark of Texas Instruments.

POSTSCRIPT is a registered trademark of Adobe Systems Incorporated. Illustrator is a trademark of Adobe Systems Incorporated.

Varityper is a registered trademark, and VT600 is a trademark, of AM International, Inc.

Simultaneously published in the United States and Canada.





# Contents



## **Preface About This Document xxi**

Intended audience xxi  
Structure of this document xxii  
Other documents you may need xxiii  
    Macintosh computer documents xxiii  
    Documents related to APPC and LU 6.2 xxiv  
Obtaining additional manuals xxv  
Conventions used in this document xxv

## **Part I Introduction**

### **Chapter 1 What Is APPC? 1-1**

The SNA network 1-3  
Functional layers 1-5  
Logical unit type 6.2 1-5  
The LU 6.2 protocol boundary 1-6  
Resource allocation 1-6  
The transaction program 1-7  
Distributed transaction processing 1-9

### **Chapter 2 What Is MacAPPC? 2-1**

Macintosh user interface 2-5  
Server-client architecture 2-5  
Security 2-7  
Transmission media 2-7  
MacAPPC drivers 2-8  
    MacAPPC Conversation Driver 2-8  
    MacAPPC Control Operator Driver 2-9  
    MacAPPC Node Operator Driver 2-9  
    MacAPPC Transaction Program Driver 2-9  
MacAPPC server 2-10  
MacAPPC Chooser device 2-10  
MacAPPC Configuration program 2-11  
MacAPPC Administration program 2-11  
MacAPPC software and server relationship 2-11



## **Part II MacAPPC Programmer's Reference**

### **Chapter 3 MacAPPC Drivers 3-1**

- Using MacAPPC drivers 3-2
- Synchronous and asynchronous execution 3-4
- MacAPPC driver parameter block 3-4
  - qLink, qType, and the I/O fields 3-5
  - appcRefNum 3-6
  - appcOpCode 3-6
  - appcHiResult and appcLoResult 3-8
  - appcConvState 3-8
  - appcUserRef 3-8
- MacAPPC driver control blocks 3-8
  - Transaction Program Control Block (TPCB) 3-8
  - Conversation Control Block (CVCB) 3-9
  - PIP buffer 3-9
  - Mapped conversation buffer 3-9
- MacAPPC driver constants 3-10
- MacAPPC driver IDs 3-11
  - Program ID 3-11
  - Conversation ID 3-11
  - Session ID 3-11
- Executing a MacAPPC driver routine 3-11
- MacAPPC driver conventions 3-12

### **Chapter 4 MacAPPC Conversation Driver 4-1**

- Using the MacAPPC Conversation Driver 4-2
  - Buffering 4-2
  - Conversation states 4-3
  - Data mapping 4-5
  - Writing a mapping procedure 4-5
    - Mapping parameter block 4-5
    - Default mapping procedure 4-7
- MacAPPC conversation routines 4-8
- Mapped conversation routines 4-8
  - MCAAllocate 4-9
  - MCConfirm 4-13
  - MCConfirmed 4-14
  - MCDeallocate 4-15
  - MCFlush 4-17
  - MCGetAttributes 4-18
  - MCPPostOnReceipt 4-21
  - MCPPrepareToReceive 4-22
  - MCReceiveAndWait 4-23
  - MCReceiveImmediate 4-25
  - MCRequestToSend 4-28
  - MCSendData 4-29
  - MCSendError 4-31
  - MCTest 4-32
- Type-independent conversation routines 4-33
  - CVBackout 4-34
  - CVGetType 4-35
  - CVSyncPoint 4-36



CVWait	4-37
Basic conversation routines	4-38
BCAllocate	4-39
BCConfirm	4-43
BCConfirmed	4-44
BCDeallocate	4-45
BCFlush	4-47
BCGetAttributes	4-48
BCPostOnReceipt	4-51
BCPrepareToReceive	4-52
BCReceiveAndWait	4-53
BCReceiveImmediate	4-55
BCRequestToSend	4-57
BCSendData	4-58
BCSendError	4-60
BCTest	4-62
Summary of the MacAPPC Conversation Driver	4-63
Constants	4-63
Data types	4-65
Mapping parameter block	4-66
Mapped routines	4-67
Type-independent routines	4-72
Basic conversation routines	4-73

## **Chapter 5 MacAPPC Control Operator Driver 5-1**

Using the MacAPPC Control Operator Driver	5-2
Security	5-2
MacAPPC control operator routines	5-3
Control operator CNOS routines	5-3
COChangeSessionLimit	5-4
COInitializeSessionLimit	5-6
COProcessSessionLimit	5-8
COResetSessionLimit	5-9
Control operator session control routines	5-11
COActivateSession	5-12
CODEactivateSession	5-13
Control operator LU definition routines	5-14
CODEfineLocalLU	5-15
CODEfineMode	5-18
CODEfineRemoteLU	5-22
CODEfineTP	5-25
CODElete	5-30
CODisplayLocalLU	5-31
CODisplayMode	5-33
CODisplayRemoteLU	5-37
CODisplaySession	5-39
CODisplayTP	5-41
Summary of the MacAPPC Control Operator Driver	5-45
Constants	5-45
Data Types	5-47
CNOS routines	5-49
Session control routines	5-51
LU definition routines	5-52



## **Chapter 6 MacAPPC Node Operator Driver 6-1**

Using the MacAPPC Node Operator Driver 6-2

MacAPPC node operator routines 6-2

Node operator node control routines 6-2

NOActivateLine 6-3

NOActivateLU 6-4

NOActivateNode 6-5

NOActivateStation 6-6

NODEactivateLine 6-7

NODEactivateLU 6-8

NODEactivateNode 6-9

NODEactivateStation 6-10

Node operator node message routines 6-11

NODefineMessageQueue 6-12

NODisplayMessage 6-14

NODisplayMessageQueue 6-15

Node operator node definition routines 6-17

NODefineCP 6-18

NODefineLine 6-20

NODefineNode 6-23

NODefineStation 6-25

NODElete 6-27

NODisplayCP 6-28

NODisplayLine 6-29

NODisplayNode 6-32

NODisplayStation 6-33

Summary of the MacAPPC Node Operator Driver 6-35

Constants 6-35

Data types 6-37

Node control routines 6-39

Node message routines 6-41

Node definition routines 6-42

## **Chapter 7 MacAPPC Transaction Program Driver 7-1**

Using the MacAPPC transaction Program Driver 7-2

Getting the currently selected MacAPPC server 7-2

Attaching and its implications 7-3

MacAPPC transaction program routines 7-4

Transaction program connection routines 7-4

TPAttach 7-5

TPDetach 7-8

Transaction program utility routines 7-9

TPAsciiToEbcdic 7-10

TPEbcdicToAscii 7-11

Summary of the MacAPPC Transaction Program Driver 7-12

Constants 7-12

Data Types 7-13

Connection routines 7-14

Utility routines 7-15



## **Chapter 8 MacAPPC Example TP 8-1**

### **Part III MacAPPC User's Guide**

## **Chapter 9 Installation 9-1**

### **Hardware 9-2**

- Client computer 9-2
- Server computer 9-2
- Communications card 9-2

### **Software 9-3**

- Client computer 9-4
- Server computer 9-4
- Configuration program 9-5
- Administration program 9-5

## **Chapter 10 Selecting a MacAPPC Server 10-1**

## **Chapter 11 The MacAPPC Configuration Program 11-1**

### **The Configuration program menu bar 11-3**

### **Conventions used in the Configuration program 11-4**

- Screen and key conventions 11-4
- Character type conventions 11-5

### **Creating a configuration file 11-5**

#### **Local node section 11-6**

- Exchange ID 11-6
- Access Type 11-6
- Monitor Timer 11-6

#### **Partner node section 11-7**

### **Creating network components 11-7**

#### **Creating local LUs 11-8**

- Name (Local LU name) 11-8
- LU ID 11-8
- Max Sess (Maximum number of sessions) 11-8
- Max TPs (Maximum number of TPs) 11-9
- User ID and password 11-9
- Profiles 11-10

#### **Creating transaction programs 11-12**

- Name (Transaction program name) 11-12
- Conv Type (Conversation type) 11-12
- Sync Level (Synchronization level) 11-12
- Security Required 11-13
- User ID 11-14
- Profile 11-14

#### **Creating Lines 11-16**

- Name (Line Name) 11-17

#### **Creating partners 11-18**

- Name (Partner name) 11-18
- Exch ID (Exchange ID) or CPU ID 11-18
- Exch ID (Exchange ID) 11-18
- CPU ID 11-18



ALS Address (Adjacent-link-station address)	11-19
Creating remote LUs	11-20
Name (Remote LU name)	11-20
Parallel Sessions	11-20
Creating modes	11-22
Name (Mode name)	11-22
Sync level (Synchronization level)	11-22
Max Sessions (Maximum number of sessions)	11-22
Min 1st Spkrs (Minimum number of first speakers)	11-22
PB Sessions (Number of prebound sessions)	11-23
Editing network components	11-24
Editing a local LU	11-24
Local LU	11-25
LU ID	11-25
Net Name (Local LU network name)	11-25
Net Qual (Local LU network qualifier name)	11-25
Max Sess (Maximum number of sessions)	11-25
LU Security	11-26
Wait Time	11-26
Max TPs (Maximum number of transaction programs)	11-26
User IDs	11-26
Profiles	11-26
Password	11-27
Editing a transaction program	11-27
TP Name	11-27
Local LU	11-27
Net Name (Transaction program network name)	11-27
Status	11-28
Conv Type (Conversation type)	11-28
Sync Level (Synchronization level)	11-28
PIP (Program initialization parameters)	11-28
PIP Count	11-28
PIP Check	11-28
Data Mapping	11-29
FMH Data	11-29
Privilege	11-29
LUW (logical unit of work)	11-29
Security Required (Security level that is required)	11-29
User ID	11-30
Profile	11-30
Editing a line	11-31
Line Name	11-32
Line Type	11-32
Line Number	11-32
Role Type	11-32
Connect Type (Connection type)	11-32
Max BTU (Maximum basic transmission unit length)	11-32
Line Speed	11-32
Max Retries (Maximum number of retries)	11-32
Idle Time	11-33
NP Recv Time (Nonproductive receive time)	11-33
Max I-Frames (Maximum number of I-frames)	11-33
NRZI Support	11-33



Duplex Type	11-33
Editing a partner	11-33
Partner Name	11-34
Line Name	11-34
Exch ID (Exchange ID) or CPU ID	11-34
Exch ID	11-34
CPU ID	11-34
ALS Address	11-35
Phone Number	11-35
Editing a remote LU	11-35
Remote LU	11-35
Local LU	11-35
Net Name (Remote LU network name)	11-36
Net Qual (Remote LU network qualifier)	11-36
CP Name (Control point or partner name)	11-36
Init Q Req (Queue session-initiation requests)	11-36
Parallel Sess (Parallel sessions)	11-36
CNOS ALS (CNOS adjacent-link-station name)	11-36
Password	11-36
Lcl Sec (Local security)	11-37
Editing a mode	11-37
Mode Name	11-37
Local LU	11-37
Remote LU	11-37
Adj Station (Adjacent station or partner name)	11-38
Send Pacing	11-38
Recv Pacing (Receive pacing)	11-38
Max RU UB (Maximum request/response unit upper bound)	11-38
Max RU LB (Maximum request/response unit lower bound)	11-38
Sync Level (Synchronization level)	11-38
Session Reinit (Session reinitiation)	11-38
Max Sessions (Maximum number of sessions)	11-39
Min 1st Spkrs (Minimum number of first speakers)	11-39
PB Sessions (Number of prebound sessions)	11-39
Queue Binds	11-39
Blank Mode	11-39
Editing defaults	11-39
Node	11-40
Local LU	11-40
TP	11-41
Line	11-41
Partner	11-42
Remote LU	11-42
Mode	11-43
Deleting network components	11-43
Printing a configuration file	11-43



## Chapter 12 The MacAPPC Administration Program 12-1

- The Administration program menu bar 12-2
- Conventions used in the Administration program 12-3
  - Special cursor 12-3
  - Network display control 12-3
  - Severity control 12-3
- Starting a MacAPPC server 12-4
  - Server Name 12-5
  - Memory Size 12-5
  - Slot 12-5
- Displaying network components and sessions 12-5
  - Exchange ID 12-6
  - Access Type 12-6
  - Monitor Timer 12-6
- Displaying a local LU 12-9
  - Local LU 12-10
  - LU ID 12-10
  - Net Name (Local LU network name) 12-10
  - Net Qual (Local LU network qualifier) 12-10
  - LU Sec (LU security) 12-10
  - Max Sess (Maximum number of sessions) 12-10
  - Act Sess (Number of active sessions) 12-10
  - Wait Time 12-10
  - Max TPs (Maximum transaction programs) 12-11
  - User IDs 12-11
  - Profiles 12-11
  - Transaction Programs 12-11
- Displaying a transaction program 12-11
  - TP Name 12-12
  - Local LU 12-12
  - Net Name (Transaction program network name) 12-12
  - Status 12-12
  - Conv Type (Conversation type) 12-12
  - Sync Level (Synchronization level) 12-12
  - PIP (Program initialization parameters) 12-12
  - PIP Count 12-12
  - PIP Check 12-13
  - Data Mapping 12-13
  - FMH Data 12-13
  - LUW (Logical unit of work) 12-13
  - Privilege 12-13
  - Security Required 12-13
  - User IDs 12-14
  - Profiles 12-14
- Displaying a line 12-14
  - Line Name 12-14
  - Line Type 12-15
  - Line Status 12-15
  - Line Number 12-15
  - Role Type 12-15
  - Connect Type 12-15
  - Max BTU (Maximum basic transmission unit length) 12-15
  - Line Speed 12-15
  - Max Retries (Maximum number of retries) 12-15



- Idle Time 12-16
- NP Recv Time (Nonproductive receive time) 12-16
- Max I-Frames (Maximum number of I-frames) 12-16
- NRZI Support 12-16
- Duplex Type 12-16
- Displaying the station and control point 12-16
  - Station Name 12-17
  - Line Name 12-17
  - Status 12-17
  - ALS Address (Adjacent-link-station address) 12-17
  - Phone Number 12-17
  - CP Name (Control point name) 12-17
  - Exch ID (Exchange ID) or CPU ID 12-17
  - Exch ID 12-17
  - CPU ID 12-18
- Displaying a remote LU 12-18
  - Remote LU 12-18
  - Local LU 12-18
  - Net Name (Remote LU network name) 12-18
  - Net Qual (Remote LU network qualifier) 12-18
  - CP Name (Control point name) 12-18
  - Init Q Req (Queue session-initiation requests) 12-19
  - Parallel Sess (Parallel sessions) 12-19
  - CNOS Name 12-19
  - Password 12-19
  - Lcl Sec (Local security) 12-19
  - Rmt Sec (Remote security) 12-19
- Displaying a mode 12-20
  - Mode Name 12-20
  - Local LU 12-20
  - Remote LU 12-20
  - Adj Station (Adjacent station name) 12-20
  - Sync Level (Synchronization level) 12-20
  - PB Sessions (Number of prebound sessions) 12-20
  - Max Sessions (Maximum number of sessions) 12-21
  - Min 1st Spkrs (Minimum number of first speakers) 12-21
  - Min Bidders (Minimum number of bidders) 12-21
  - Send Pacing 12-21
  - Recv Pacing (Receive pacing) 12-21
  - Max RU UB (Maximum request/response unit upper bound) 12-21
  - Max RU LB (Maximum request/response unit lower bound) 12-21
  - Term Count (Termination count) 12-21
  - Session Reinit (Session reinitiation) 12-22
  - Queue Binds 12-22
  - Blank Mode 12-22
  - Drain Local 12-22
  - Drain Remote 12-22
- Displaying a session 12-22
  - Session ID 12-23
  - Polar Type (Polarity type) 12-23
  - Conv ID (Conversation ID) 12-23
  - Prog ID (Program ID) 12-23



Managing network components and sessions	12-23
Updating the server window	12-23
Starting and stopping CNOS	12-24
Activating network components and sessions	12-25
Order of activation	12-25
Lines	12-25
Stations	12-25
Local LUs	12-26
Remote LUs	12-26
Modes	12-26
Sessions	12-26
Deactivating network components and sessions	12-26
Order of deactivation	12-27
Sessions	12-27
Modes	12-27
Local LUs	12-28
Stations	12-29
Lines	12-29
Logging	12-29
Log settings options	12-29
Class	12-30
Type	12-30
Severity	12-30
Check for message every _ seconds	12-30
Log Window	12-31
Stopping a MacAPPC server	12-31

## **Appendix A MacAPPC Interface File A-1**

## **Appendix B MacAPPC Errors File B-1**

## **Appendix C MacAPPC Result Codes C-1**

MacAPPC result codes	C-2
Major Code 00—noErr: Function completed normally	C-2
Major Code 01—usageErr: Function aborted, usage error	C-2
Major Code 02—badComplEr: Function aborted, bad completion	C-13
Major Code 03—stateErr: Function aborted, state error	C-13
Major Code 05—allocErr: Function aborted, allocation error	C-13
Major Code 07—progErr: Program error	C-14
Major Code 09—deallocErr: Deallocated	C-15
Major Code 10—ctlOpErr: Control operator error	C-15
Major Code 11—nodeOpErr: Node operator error	C-16

## **Appendix D MacAPPC Routine Mapping D-1**

Conversation routine mapping	D-2
Control operator routine mapping	D-3
Node operator routine mapping	D-3
Transaction program routine mapping	D-4



## **Appendix E MacAPPC Conversation Parameter Mapping E-1**

MC\_ALLOCATE is MCAAllocate E-2  
MC\_CONFIRM is MCConfirm E-2  
MC\_CONFIRMED is MCConfirmed E-2  
MC\_DEALLOCATE is MCDeallocate E-3  
MC\_FLUSH is MCFlush E-3  
MC\_GET\_ATTRIBUTES is MCGetAttributes E-3  
MC\_POST\_ON\_RECEIPT is MCPostOnReceipt E-4  
MC\_PREPARE\_TO\_RECEIVE is MCPrepareToReceive E-4  
MC\_RECEIVE\_AND\_WAIT is MCReceiveAndWait E-4  
MC\_RECEIVE\_IMMEDIATE is MCReceiveImmediate E-5  
MC\_REQUEST\_TO\_SEND is MCRequestToSend E-5  
MC\_SEND\_DATA is MCSendData E-5  
MC\_SEND\_ERROR is MCSendError E-6  
MC\_TEST is MCTest E-6  
BACKOUT is CVBackout E-6  
GET\_TYPE is CVGetType E-6  
SYNCPT is CVSynchPoint E-7  
WAIT is CVWait E-7  
ALLOCATE is BCAAllocate E-7  
CONFIRM is BCConfirm E-8  
CONFIRMED is BCConfirmed E-8  
DEALLOCATE is BCDeallocate E-8  
FLUSH is BCFlush E-8  
GET\_ATTRIBUTES is BCGetAttributes E-9  
POST\_ON\_RECEIPT is BCPostOnReceipt E-9  
PREPARE\_TO\_RECEIVE is BCPrepareToReceive E-9  
RECEIVE\_AND\_WAIT is BCReceiveAndWait E-10  
RECEIVE\_IMMEDIATE is BCReceiveImmediate E-10  
REQUEST\_TO\_SEND is BCRequestToSend E-10  
SEND\_DATA is BCSendData E-11  
SEND\_ERROR is BCSendError E-11  
TEST is BCTest E-11

## **Appendix F MacAPPC Control Operator Parameter Mapping F-1**

CHANGE\_SESSION\_LIMIT is COChangeSessionLimit F-2  
INITIALIZE\_SESSION\_LIMIT is COInitializeSessionLimit F-2  
RESET\_SESSION\_LIMIT is COResetSessionLimit F-2  
PROCESS\_SESSION\_LIMIT is COProcessSessionLimit F-3  
ACTIVATE\_SESSION is COActivateSession F-3  
DEACTIVATE\_SESSION is CODEactivateSession F-3  
DEFINE\_LOCAL\_LU is CODEfineLocalLU F-3  
DEFINE\_REMOTE\_LU is CODEfineRemoteLU F-4  
DEFINE\_MODE is CODEfineMode F-4  
DEFINE\_TP is CODEfineTP F-5  
DISPLAY\_LOCAL\_LU is CODisplayLocalLU F-6  
DISPLAY\_REMOTE\_LU is CODisplayRemoteLU F-6  
DISPLAY\_MODE is CODisplayMode F-7  
DISPLAY\_TP is CODisplayTP F-7  
DELETE is CODElete F-8



## **Appendix G MacAPPC Result Codes Mapping G-1**

Conversation return codes G-2

Control operator return codes G-3

## **Appendix H HyperCard APPC H-1**

Setup H-2

Physical requirements H-2

Software requirements H-2

Starting the MacAPPC server H-2

Overview H-3

Introduction H-3

HyperCard APPC Application Programming Interface  
(API) H-3

MacAPPC Lab H-4

MacAPPC routine cards H-6

Supplied Values parameters H-7

Returned Values parameters H-7

Other elements H-9

Lab Help H-9

Sample Application H-11

APPC Mail Configuration H-12

Postmaster H-13

Mailbox H-14

Sample session: Stepping through a conversation H-15

Developing HyperCard APPC applications H-20

APPC XCMD H-20

get62Srvr XCMD H-21

xConst XFCN H-22

errStr XCMD H-22

HyperCard APPC scripts and handlers H-23

Stack script H-23

Script of the first card H-23

Routine cards' background script H-25

Scripts of the routines cards H-25

Background script of the sample application H-26

Postmaster card script H-27

Mailbox card script H-27

XData XCMDs and XFCNs H-28

xDefine H-29

xGlobal H-30

xPut H-31

xGet H-32

xLock H-32

xPtr H-32

xFill H-33

xSize H-33

xDispose H-33

xMove H-34

xResource H-34

Special parameters H-36

*fieldSpec* parameter H-36

*ptrType* parameter H-37

Sample handlers H-37

XData errors H-39



Basic data types H-40

Byte alignment of fields H-41

**Appendix I MacAPPC Option Sets I-1**

**Appendix J ASCII/EBCDIC Tables J-1**

**Appendix K Configuration Worksheets K-1**

**Glossary G-1**

**Index General Index & Index of Parameters and Constants**







---

---

## Figures and tables

### Part I Introduction

#### Chapter 1 What Is APPC? 1-1

- Figure 1-1 SNA network components 1-4
- Figure 1-2 An APPC distributed transaction 1-8
- Figure 1-3 A pair of transaction programs that share SNA resources 1-9

#### Chapter 2 What Is MacAPPC? 2-1

- Figure 2-1 A hypothetical logical network structure 2-2
- Figure 2-2 The MacAPPC environment 2-3
- Figure 2-3 The MacAPPC server-client relationship 2-6
- Figure 2-4 MacAPPC connection types 2-8
- Figure 2-5 MacAPPC interactions 2-12
- Figure 2-6 MacAPPC programs and server relationship 2-13

### Part II MacAPPC Programmer's Reference

#### Chapter 3 MacAPPC Drivers 3-1

- Table 3-1 MacAPPC drivers and the categories of routines 3-3
- Table 3-2 MacAPPC drivers and their parameter blocks 3-4

#### Chapter 4 MacAPPC Conversation Driver 4-1

- Table 4-1 States for mapped conversation routines 4-3
- Table 4-2 States for type-independent conversation routines 4-4
- Table 4-3 States for basic conversation routines 4-4

#### Chapter 7 MacAPPC Transaction Program Driver 7-1

- Table 7-1 Routines valid for different attach types 7-3

### Part III MacAPPC User's Guide

#### Chapter 9 Installation 9-1

- Table 9-1 AST-ICP communications card jumper settings 9-3
- Table 9-2 MacAPPC User disk 9-3
- Table 9-3 MacAPPC System disk 9-4

#### Chapter 10 Selecting a MacAPPC Server 10-1

- Figure 10-1 The Chooser 10-2
- Figure 10-2 Selecting a MacAPPC server 10-3



## Chapter 11 The MacAPPC Configuration Program 11-1

Figure 11-1	The File menu 11-3
Figure 11-2	The Edit menu 11-3
Figure 11-3	The Create menu 11-4
Figure 11-4	Creating a new configuration file 11-5
Figure 11-5	The configuration file window 11-6
Figure 11-6	Configuration network components 11-7
Figure 11-7	Creating a local LU 11-8
Figure 11-8	The local LU window 11-9
Figure 11-9	Creating a new user ID and password for a local LU 11-9
Figure 11-10	Local LU window with new user IDs and password 11-10
Figure 11-11	Creating a new profile 11-10
Figure 11-12	Local LU user IDs, profiles, and password 11-11
Figure 11-13	The local LU in relation to other components 11-11
Figure 11-14	Creating a transaction program for a local LU 11-12
Figure 11-15	The TP window 11-13
Figure 11-16	Selecting a user ID for a TP 11-14
Figure 11-17	Selecting a profile for a TP 11-15
Figure 11-18	TP window with user ID and Profile lists 11-15
Figure 11-19	The transaction program in relation to other components 11-16
Figure 11-20	Creating a line 11-17
Figure 11-21	The line in relation to other components 11-17
Figure 11-22	Creating a partner node 11-18
Figure 11-23	The partner in relation to other components 11-19
Figure 11-24	Creating a remote LU 11-20
Figure 11-25	The remote LU in relation to other components 11-21
Figure 11-26	Creating a mode 11-22
Figure 11-27	The mode in relation to other components 11-23
Figure 11-28	The configuration file window for the example network node 11-24
Figure 11-29	Editing a local LU 11-25
Figure 11-30	Editing a user ID 11-26
Figure 11-31	Editing a profile 11-26
Figure 11-32	Editing a transaction program 11-27
Figure 11-33	Creating a user ID 11-30
Figure 11-34	Creating a profile 11-31
Figure 11-35	Editing a line 11-31
Figure 11-36	Editing a partner (control point and station) 11-34
Figure 11-37	Editing a remote LU 11-35
Figure 11-38	Editing a mode 11-37
Figure 11-39	Editing node default settings 11-40
Figure 11-40	Editing local LU default settings 11-40
Figure 11-41	Editing TP default settings 11-41
Figure 11-42	Editing line default settings 11-41
Figure 11-43	Editing partner default settings 11-42
Figure 11-44	Editing remote LU default settings 11-42
Figure 11-45	Editing mode default settings 11-43
Figure 11-46	Example of a configuration file printout 11-44
Table 11-1	Security options 11-13



## **Chapter 12 The MacAPPC Administration Program 12-1**

Figure 12-1	The File menu 12-2
Figure 12-2	The Edit menu 12-2
Figure 12-3	The Server menu 12-3
Figure 12-4	The Log menu 12-3
Figure 12-5	Selecting a configuration file 12-4
Figure 12-6	Editing server settings 12-5
Figure 12-7	The server window 12-6
Figure 12-8	Displaying components at the local LU level 12-7
Figure 12-9	Displaying the components at the remote LU level 12-8
Figure 12-10	Displaying components at the mode level 12-8
Figure 12-11	Displaying components and sessions at the session level 12-9
Figure 12-12	A local LU display window 12-10
Figure 12-13	A transaction program display window 12-11
Figure 12-14	A line display window 12-14
Figure 12-15	A station display window 12-17
Figure 12-16	A remote LU display window 12-18
Figure 12-17	A mode display window 12-20
Figure 12-18	A session display window 12-22
Figure 12-19	Activating a station 12-25
Figure 12-20	Deactivating a session 12-27
Figure 12-21	Deactivating a mode 12-28
Figure 12-22	Deactivating a station 12-29
Figure 12-23	Log settings selection 12-30
Figure 12-24	The log window 12-31
Figure 12-25	Stopping a server 12-32

## **Appendix H HyperCard APPC H-1**

Figure H-1	The HyperCard APPC stack title card H-3
Figure H-2	The navigation button pop-up menu H-4
Figure H-3	Navigation cards for the MacAPPC Lab H-5
Figure H-4	TPAttach routine card H-6
Figure H-5	Remote LU Name help field H-10
Figure H-6	Help on error result codes H-11
Figure H-7	APPC Mail title card H-12
Figure H-8	APPC Mail configuration card H-12
Figure H-9	APPC Mail Postmaster H-13
Figure H-10	APPC Mail Mailbox H-14
Table H-1	APPC major errors (APPC Hi Result) H-8
Table H-2	APPC conversation state H-8
Table H-3	HyperCard APPC record types H-24
Table H-4	XData errors H-39
Table H-5	Basic data types H-40

## **Appendix I MacAPPC Option Sets I-1**

Table I-1	Supported APPC option sets I-1
-----------	--------------------------------

## **Appendix J ASCII/EBCDIC Tables J-1**

Table J-1	ASCII to EBCDIC J-1
Table J-2	EBCDIC to ASCII J-2



## **Appendix K Configuration Worksheets K-1**

Figure K-1 MacAPPC remote configuration worksheet K-2  
Figure K-2 MacAPPC local configuration worksheet K-3





## Preface



# About This Document

This document introduces MacAPPC™, an advanced communications product that is Apple's implementation of the IBM LU 6.2 communications protocols on the Apple® Macintosh® computer. Within the design of Systems Network Architecture (SNA), LU 6.2 permits program-to-program communication on processors that operate on a peer-to-peer basis. LU 6.2 has become the IBM standard for distributed processing.

The implementation of LU 6.2 is known as *Advanced Program-to-Program Communication* (APPC), which provides interprogram communication across systems that use APPC. With APPC, transaction programs can be written to coordinate distributed processing across a network of computers and peripheral devices.

This Apple implementation of APPC is called *MacAPPC*. Programs that use MacAPPC on a Macintosh computer, or a network of Macintosh computers, are able to communicate on a peer-to-peer basis with programs on other nodes of an SNA network and perform distributed transaction processing with those programs.

With MacAPPC, a Macintosh programming environment is provided for additional extensions of the SNA design, such as SNA Distribution Services (SNADS) and Document Interchange Architecture (DIA), two major components in the SNA design of office automation systems. Both of these systems use APPC for their foundation.

---

---

### Intended audience

This document is intended for third-party software developers, MIS programmers and managers, government systems integrators, and value-added resellers—anyone who wishes to use the Apple implementation of APPC to write transaction programs in a mixed IBM-Macintosh environment.

This document provides complete descriptions of the MacAPPC components, as well as references to other source documents, to permit you to write APPC transaction programs.

This document assumes that you have extensive development experience with the Macintosh computer, or else are able to obtain such information from other documents (such documents are listed in "Other Documents You May Need," later in the Preface). It is also assumed that you are familiar with the IBM development environment. Extensive knowledge is not assumed regarding SNA or APPC.



---

---

## Structure of this document

This document is divided into three parts and several appendixes that contain the following information:

- Part I, "Introduction," explains the purpose of the MacAPPC product and briefly introduces the SNA and LU 6.2 concepts and protocols that are used by MacAPPC.
- Part II, "MacAPPC Programmer's Reference," provides a complete description of the MacAPPC driver routines and itemizes and defines each parameter for each routine. This part, intended only for programmers, is divided into a chapter that describes the general structure of the MacAPPC drivers, and individual chapters that describe the conversation routines, control operator routines, node operator routines, and transaction program routines. Within each chapter, the categories of the routines are further broken down according to function. Finally, a chapter is provided that shows a sample transaction program.
- Part III, "The MacAPPC User's Guide," is a thorough presentation of the configuration and operation of MacAPPC, as well as the functions of the server, the Chooser device, and the device drivers. Part III assumes little or no knowledge of MacAPPC; it is designed to show anyone how to configure a MacAPPC server, as long as the proper system information is provided.
- Appendix A, "MacAPPC Interface File," contains a printout of the C language interface to the MacAPPC drivers.
- Appendix B, "MacAPPC Errors File," contains a printout of the MacAPPC error values.
- Appendix C, "MacAPPC Result Codes," describes in detail the return code parameters that the MacAPPC driver issues to indicate the success or failure of a routine request.
- Appendix D, "MacAPPC Routine Mapping," provides a mapping of the IBM-defined set of LU 6.2 verbs to the MacAPPC equivalents.
- Appendix E, "MacAPPC Conversation Parameter Mapping," provides a mapping of the IBM-defined set of parameters for each LU 6.2 conversation verb to its MacAPPC equivalent.
- Appendix F, "MacAPPC Control Operator Parameter Mapping," provides a mapping of the IBM-defined set of parameters for each LU 6.2 control operator verb to its MacAPPC equivalent.
- Appendix G, "MacAPPC Result Code Mapping," provides a mapping of the LU 6.2 conversation and control operator return codes to their MacAPPC equivalents.
- Appendix H, "HyperCard APPC™" provides an introduction to a HyperCard® stack that helps you program in the MacAPPC environment.
- Appendix I, "MacAPPC Option Sets" provides a list of the LU 6.2 options supported by MacAPPC.
- Appendix J, "ASCII/EBCDIC Tables" provides tables for translating between ASCII and EBCDIC and vice versa.
- Appendix K, "MacAPPC Configuration Worksheets" provides worksheets to help during use of the MacAPPC Configuration Program.
- The glossary provides brief definitions of important MacAPPC and LU 6.2 terms.



---

---

## Other documents you may need

The following documents are either referred to in this document, or else are recommended for additional information on LU 6.2 or the Macintosh computer.

---

### Macintosh computer documents

The *Apple Technical Library*, published by Addison-Wesley, is a set of technical books from Apple Computer, Inc. It includes books that explain the hardware and software of the Macintosh family of computers. The descriptions that follow may help you decide which of the books will be most useful to you.

- ❑ *Inside Macintosh*, Volumes I, II, and III. These books cover the Macintosh Toolbox and Operating System for the original 64K Macintosh ROM, along with user interface guidelines and hardware information.
- ❑ *Inside Macintosh*, Volume IV. A delta guide (that is, it covers only what is new) for the Macintosh Plus and Macintosh 512K enhanced computers (128K ROM).
- ❑ *Inside Macintosh*, Volume V. Also a delta guide. It covers what is different about the Macintosh SE and Macintosh II computers (256K ROM).
- ❑ *Technical Introduction to the Macintosh Family*. An introduction to the hardware and software design of the Macintosh family. This book serves as a starting point for Macintosh technical documentation. It is oriented primarily toward the Macintosh Plus, Macintosh SE, and Macintosh II computers, but it also touches on earlier versions of the Macintosh where these differ from the Macintosh Plus.
- ❑ *Programmer's Introduction to the Macintosh Family*. Provides an overview of software development for the Macintosh family of computers. This book focuses on the differences between event-driven programming and more traditional programming techniques. It covers such topics as QuickDraw™ graphics, screen displays, and the Macintosh User Interface Toolbox.
- ❑ *Human Interface Guidelines: The Apple Desktop Interface*. A description of the Apple user interface for the benefit of people who want to develop applications.
- ❑ *Inside Macintosh X-Ref*. Comprehensive indexes, routine lists, and a glossary for *Inside Macintosh* and other Macintosh programming books.

Other books that may be helpful include the following, which are available from the Apple Programmer's and Developer's Association (APDA™).

- ❑ *Macintosh Programmer's Workshop Reference*. A guide to the Macintosh Programmer's Workshop (MPW™) Shell and utilities, including the resource editor (ResEdit), resource compiler (Rez), Linker, Make facility, and debugger.
- ❑ *MPW Assembler Reference*. A guide to preparing source files to be assembled by the Macintosh Programmer's Workshop Assembler.
- ❑ *MPW Pascal Reference*. This manual provides information about the MPW Pascal language and the use of the MPW Pascal programming system.
- ❑ *MPW C Reference*. This manual tells how to write C programs that you can link with programs written in MPW Pascal.
- ❑ *Macintosh Coprocessor Platform Developer's Guide*. This manual is the guide to the real-time multi-tasking operating system (MCP operating system) for the smart card on which MacAPPC runs.



The Apple Programmers and Developers Association (APDA™) provides a wide range of technical products and documentation, from Apple and other suppliers, for programmers and developers who work on Apple equipment. For information about APDA, contact:

Apple Programmers and Developers Association  
Apple Computer, Inc.  
20525 Mariani Avenue, Mailstop 33-G  
Cupertino, CA 95014-6299  
1-800-282-APDA (1-800-282-2732)  
Fax: 408-562-3971  
Telex: 171-576  
AppleLink®: APDA

If you plan to develop hardware or software products for sale through retail channels, you can get valuable support from Apple Developer Programs. Write to:

Apple Developer Programs  
Apple Computer, Inc.  
20525 Mariani Avenue, Mailstop 51-W  
Cupertino, CA 95014-6299

---

## Documents related to APPC and LU 6.2

The following is a list of IBM documents that are necessary for anyone writing transaction programs that use LU 6.2 protocols.

- *SNA Format and Protocol Reference Manual for LU Type 6.2* (GC30-3084). This manual, often referred to as the "FAP manual" or "FAPL," describes in detail the APPC functions, as well as underlying LU 6.2 aspects that are invisible to transaction programs.
- *SNA Format and Protocol Reference Manual for Type 2.1 Nodes* (SC30-3422). The PU 2.1 manual provides detailed information on the PU 2.1 node similar to that contained in the LU 6.2 FAPL.
- *SNA Transaction Programmers Reference Manual for LU Type 6.2* (SC30-3269). This manual, often referred to as the "TPRM manual" or "TPRM," provides a formal description of the syntax used to define the protocol boundary, as well as the view of LU 6.2 as seen from the perspective of a transaction program.

The following book may be useful to transaction program writers, or anyone else interested in an overview of the LU 6.2 protocols.

- *An Introduction to Advanced Program-to-Program Communication (APPC)* (GG24-1584-01). This manual is an introduction to APPC, including overviews of both the FAPL and the TPRM, as well as product implementations.



---

---

## Obtaining additional manuals

Additional copies of this manual can be obtained through APDA.

---

---

## Conventions used In this document

In this document, terms are printed in **boldface** when they are introduced. These terms are also included in the glossary.

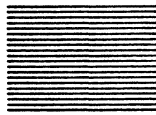
Program listings, and text that is a term used by MacAPPC, is presented in `Courier` typeface.

The term *verb* is used within APPC to describe an APPC routine. In this document, the more familiar Macintosh term *routine* is used throughout.









## **Part I**



# **Introduction**

This part consists of two chapters that introduce the user to Systems Network Architecture (SNA), Advanced Program-to-Program Communications (APPC), and MacAPPC, Apple's implementation of APPC on the Apple Macintosh computer.

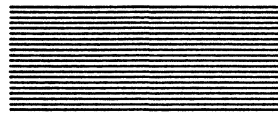
Chapter 1 provides a brief introduction to SNA and APPC and describes functional layers, logical unit type 6.2, the LU 6.2 protocol boundary, resource allocation, the transaction program, and distributed transaction processing.

Chapter 2 introduces the user to MacAPPC and describes the Macintosh user interface, server-client architecture, security, transmission media, MacAPPC drivers, MacAPPC Chooser device, MacAPPC Configuration program, MacAPPC Administration program, and the MacAPPC software and server relationship.

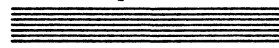








## **Chapter 1**



### **What Is APPC?**



In 1974, IBM introduced its **Systems Network Architecture (SNA)** for the purpose of uniting several generations of mainframe computers, operating systems, peripheral devices, and telecommunications systems within a single global communications architecture. Prior to SNA, these components were often incompatible; with SNA, the unification of heterogeneous hardware platforms and operating systems became possible for the first time.

In 1985, IBM announced **Advanced Program-to-Program Communication (APPC)**, an enhancement to SNA. APPC enhances SNA by simplifying network design, improving data handling, reducing network overhead, and providing a single standard for other vendors. This is accomplished by breaking away from the hierarchical, master-slave relationship of mainframe to terminal that was an integral part of SNA.

APPC provides direct program-to-program links and true program-to-program connectivity. With APPC, intelligent devices such as personal computers and workstations are able to communicate with each other and initiate tasks on a network with authority equal to that of the mainframe. In this fashion, APPC permits true distributed processing; in fact, APPC is now IBM's single strategic architecture for distributed transaction processing.

The equality provided by APPC is made possible by the design of its software components, logical unit (LU) type 6.2 and physical unit (PU) type 2.1. LU type 6.2 provides program-to-program communication between transaction programs (programs that provide transaction processing). PU type 2.1 provides program-to-program communication between hardware units. These components are designed to make use of the tremendous power of personal computers that has developed since the advent of SNA.

Distributed processing allows the intelligence in a network to be shared among the network's participants. By distributing the intelligence, costs can be reduced and network traffic can be relieved. With APPC established as a standard interface, you may have products of various vendors and still be able to unify your heterogeneous network without duplication of equipment.

Distributed error recovery is part of the control of resources that are distributed across a network. Distributed error recovery provides for the restoring of resources to their original states when errors occur.

The management and allocation of resources across the network is a central function of APPC. An LU may schedule the allocation of resources and provide necessary services.

A network that implements APPC is able to participate in the following functional benefits:

- ☐ distributed processing of transactions across the network
- ☐ distributed error recovery
- ☐ distributed resource management across the network in distributed transaction processing



With APPC, a network can fully exploit the capabilities of extensions to SNA that provide office automation architectures, such as SNA Distribution Services (SNADS) and Document Interchange Architecture (DIA). Applications already exist that make use of these designed extensions, such as the Distributed Office Support System (DISOSS).

---

---

## The SNA network

An SNA network is divided into physical and logical components. The physical network consists of actual processors, called **nodes**, and data links between the nodes. The logical network consists of a set of software components called **network addressable units (NAUs)**, including logical units (LUs), physical units (PUs), and system services control points (SSCP). These NAUs are interconnected by a path control network consisting of the path control, data link control, and physical layers.

The logical connection between two NAUs is a **session**; although several types of sessions exist, the end user is aware of only one type, which is LU to LU. When two LUs establish a session, information flows between them by means of a data stream. Over the session, LUs exchange information by means of message units called **request/response units (RUs)**.

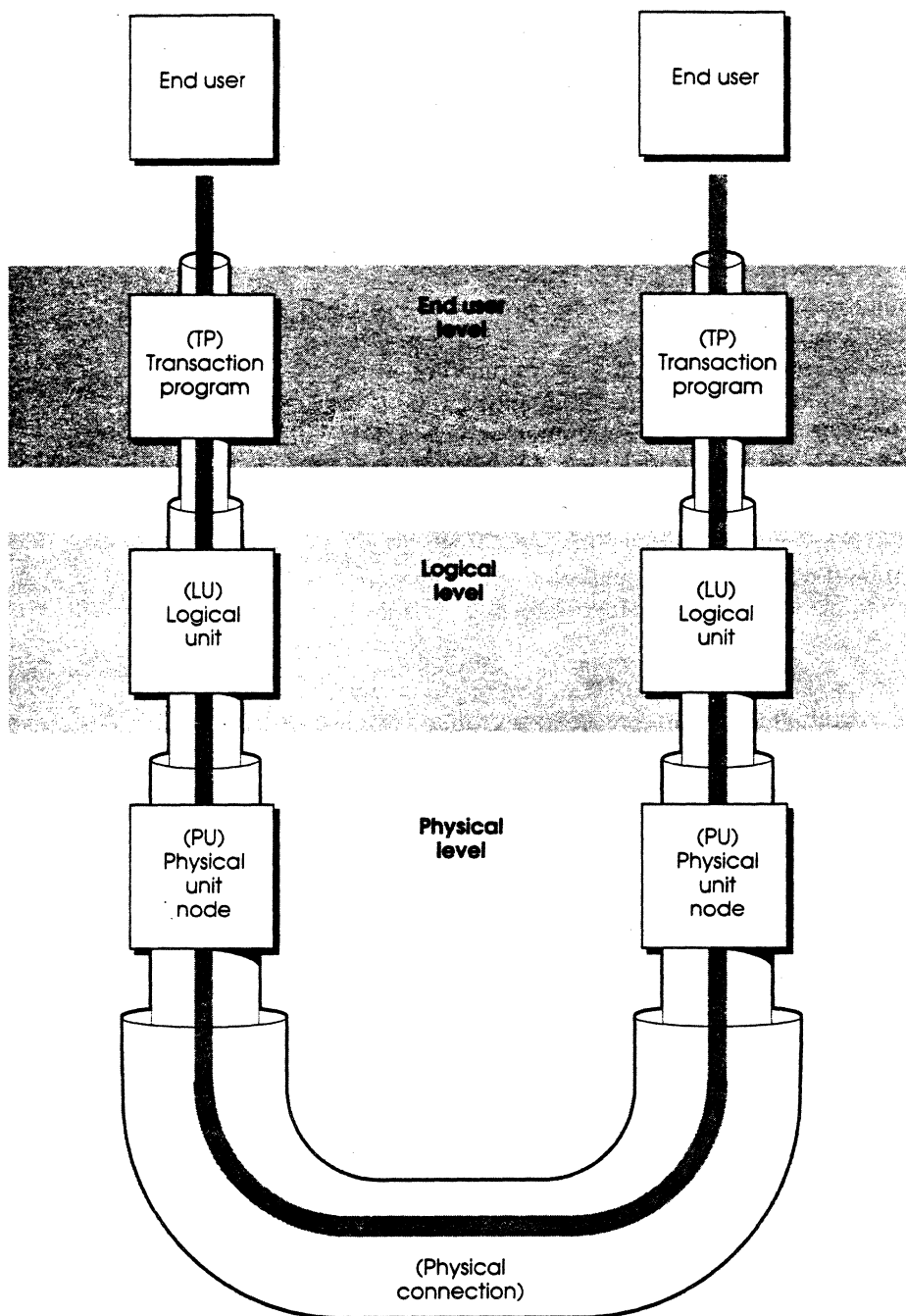
Sessions are established when one LU sends another LU an SNA request known as a **bind**. Traditionally, the bind determines a **primary** and a **secondary** for the session, usually with the primary LU residing in the host, and the secondary LU residing in a terminal or other device. With LU type 6.2, either LU may serve as the primary unit; the decision as to which is primary and which is secondary is made during session initiation.

The LU provides a connection into SNA for the end user, which may either be an individual or a transaction program. The LUs provide protocols that allow end users to communicate with each other and with other NAUs in the network. An LU can be associated with more than one network address; this allows two LUs to form multiple, concurrently active sessions between them.

There may be a one-to-one relationship between end users and LUs, but in general, this is not necessary. The association between end users and the set of LUs is a function of the implementation design.

Figure 1-1 shows a schematic outline of network components of a configuration. The illustration provides a visual representation of the SNA network components and their relationship to each other. The physical level consists of PU nodes linked by a physical connection. The logical level consists of LUs that link the PUs and the transaction programs (TPs). The end user level consists of TPs that communicate with other TPs using LUs. The mode, not shown in Figure 1-1, is a single-session connection or a group of parallel sessions having similar session-level parameters and path-control characteristics.





**Figure 1-1**  
SNA network components



---

---

## Functional layers

The SNA network is divided into layers, each corresponding to a particular set of network functions. Information is passed through the layers of the network by means of message units (the RU, for example, is one category of message unit).

As a message is passed up and down the SNA functional layers, each layer performs a set of control functions and may add control information to the message in the form of message headers. These headers do not change the information in the message, but merely communicate with the next layer of SNA to ensure that the message reaches its next destination and is properly understood. As the message passes through layers, the information headers that were added at one end of the network are stripped off and read by the receiving end of the network. When the message reaches its destination, it is back to its original form.

SNA carefully defines the interaction between network layers, including the parameters that are included in the message headers. The logical communication that takes place between corresponding layers is defined as **program-to-program communication**. It is the program-to-program protocol specifications that guarantee communication between nodes that are of different types, are from different hardware manufacturers, and have different software components.

The layers that make up an SNA network are grouped into two functional categories:

- ❑ NAU services, consisting of the transmission control layer, data flow control layer, and function management layer
- ❑ path-control network services, consisting of the path-control and data-link control layers of the network

As protocols are discussed and explained in this document, their functionality is described in terms of their relationship to either NAU services or path-control network services.

---

---

## Logical unit type 6.2

LU 6.2 is a particular type of SNA logical unit. Several LU types are defined in SNA; each type provides transmission capabilities and a set of services, such as resource allocation, for a particular end user. Each hardware or software component on an SNA network is assigned to an LU type; for example, LU types are defined for various terminal types and printers. LU 6.2 is the LU type that provides the basis for APPC. Prior to LU 6.2, remote devices were allowed only limited capability, such as terminal emulation—that is, they were unable to participate in the processing of actual work. Intelligent tasks were left to the mainframe host or front-end processor.

With LU 6.2, a program residing in one device is able to access a program in a remote device. The set of services that allows programs using LU 6.2 to communicate with each other is known as **interprogram communication**. Because LU 6.2 provides the basis for this communication, the term *LU 6.2* is often used as a synonym for *APPC*. The purpose of APPC is to facilitate the development of distributed applications by providing a set of verbs that are defined by the SNA architecture. Each routine corresponds to an LU 6.2 service requested by a transaction program.



Interprogram communication permits distribution of the processing of a transaction among multiple programs within a network. The programs coordinate the distributed processing by exchanging control information or data. The LU 6.2 **protocol boundary** provides the structure for programs to communicate with one another in order to process a transaction. The LU 6.2 protocol boundary is described in the next section.

The capabilities provided by APPC are not all derived from the logical unit—much of the power is provided by the LU's physical counterpart, the PU (physical unit). The physical unit provides services needed to access and manage hardware units on the network. Each PU type is assigned a number: for example, PU type 5 defines a mainframe, PU 4 indicates a controller, and PU 2 defines a user-programmable terminal node. The counterpart of LU 6.2 is PU 2.1, an enhancement of PU 2.0 that provides program-to-program communication between PUs.

Unlike other LU types, LU type 6.2 is able to handle various data streams. A **data stream** is defined as the characters and control codes that are passed between logical units. For earlier LU types, the data stream was defined—for example, LU type 2 requires a 3270 data stream. A session of LU type 6 (of which LU 6.2 is a subset) can use any data stream, thus making it compatible with a wide range of devices.

LU 6.2 provides a connection for its end user to the path-control network. The end user may be an individual, accessing information via a terminal; in distributed processing, the end user is a particular kind of application program, known as a *transaction program (TP)*, that performs a certain task (such as updating a database).

---

---

## The LU 6.2 protocol boundary

The LU 6.2 protocol boundary is defined by the set of verbs and parameters that make up the transaction program's logical interface to an SNA network (for MacAPPC routines and parameters, see Part II, "MacAPPC Programmer's Reference"). The protocol boundary is generic in the sense that it provides a syntactical representation of the functions common to all products that implement LU 6.2. The value of a generic description is that a program designer may plan an application that spans many different products using a single generic interface, and then map the design to the individual product-dependent interfaces.

The verbs (called *routines* in MacAPPC) that make up the protocol boundary may be used to invoke services (such as synchronization services) by application subsystems (such as CICS/VS). A subsystem that has its own application programming interface may use another language or syntax to represent the APPC verbs to its application programs. For example, a CICS programmer could use CICS commands that would be translated into LU 6.2 routines. These commands could be communicated to another CICS system that would translate the LU 6.2 verbs back into CICS commands—the process is transparent to the end user.



---

---

## Resource allocation

In order to assist the transaction program in fulfilling its task, the LU makes a set of resources available to it. Some resources are **local** to a transaction program, that is, attached to the same LU as the program. Other resources are **remote**, which means that they are attached to other LUs. Remote is defined in terms of the logical configuration of the network; the LUs can be within the same physical node.

Resource allocation and control is a central function of APPC. Within APPC, programs can ask the LU for access to a resource. The LU schedules allocation of resources, and handles much of the overhead—for example, it creates new copies of logical resources, such as sessions, when necessary. The LU provides resource control in order to ensure integrity of the program's access to the resource.

---

---

## The transaction program

A **transaction program (TP)** is a program that is executed by or within APPC and performs services related to the processing of a transaction. A transaction involves a specific set of input data and triggers the execution of a specific process or job. An example is the entry of a bank customer's withdrawal or deposit and the updating of the customer's balance.

The TP may be

- ❑ an application program that processes a transaction
- ❑ an application that is one of several programs that make up a transaction processing application
- ❑ a system program that performs system services for an application program processing a transaction

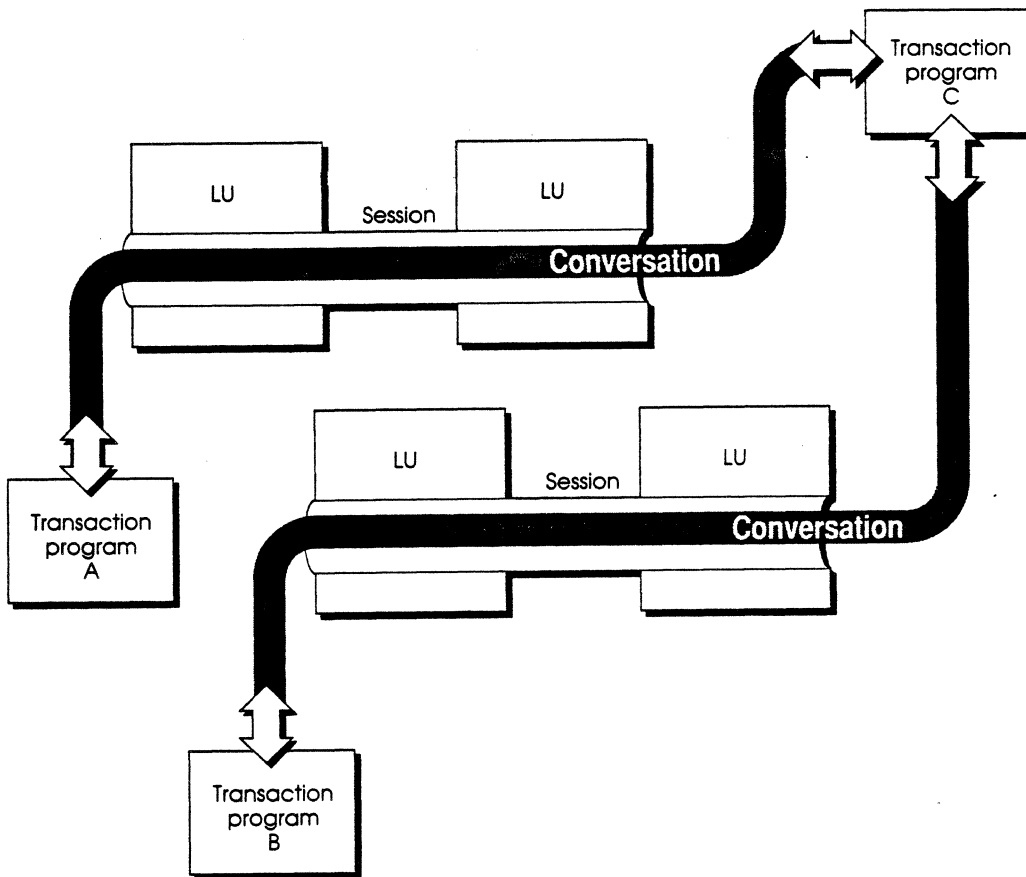
The purpose of APPC is to provide programmers with the means of writing TPs that can access nodes across the network, regardless of the brand or type of equipment involved. This means that protocols unique to each particular computer or peripheral device need not be hard-coded into a transaction program. Instead, the TP can search for and access information from any node, without regard for node model or manufacturer.

The logical state that exists between two network addressable units to support a succession of transmissions between the units is called a **session**. Two LUs may connect to each other by one LU-LU session, called a **single session**, or connect to each other by multiple LU-LU sessions, called **parallel sessions**. At the initiation of a single or parallel LU-LU session, only one LU is the contention winner; the other is the contention loser. This contention between the two LUs is called **polarity**.

The logical connection between a pair of transaction programs is called a **conversation**. A TP initiates a conversation with its partner with the assistance of the LUs. While a conversation is active, it has exclusive use of a session, but successive conversations may use the same session.



Conversations connect TPs in pairs; however, any TPs directly or indirectly connected to each other by conversations are participating in the same distributed transaction. For example, in Figure 1-2, if TP A and TP C are connected by a conversation, and, concurrently, TP B and TP C are connected by a conversation, then TPs A, B, and C are all participating in the same distributed transaction.



**Figure 1-2**  
An APPC distributed transaction

In the event of a conversation failure, the session can remain active. The conversation is seen as a single unit of work, and may be precisely delimited by a data flow control service called **bracketing**. By keeping the conversation intact as a unit, bracketing helps guarantee data integrity.

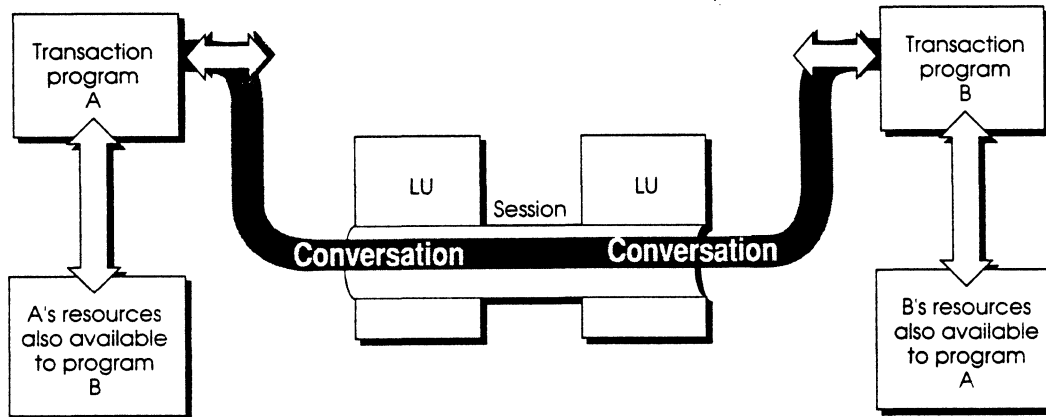


---

---

## Distributed transaction processing

Distributed processing involves two or more programs, usually at different systems, cooperating to carry out some processing function. Distributed processing of a transaction within an APPC network occurs when transaction programs communicate by exchanging information over the sessions between their LUs. Figure 1-3 illustrates the connection of two programs to APPC resources, including a session between their LUs.



**Figure 1-3**  
A pair of transaction programs that share SNA resources

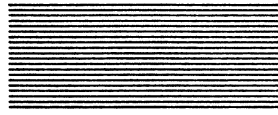
The “other resources” shown in the figure may include other sessions as well as local files and devices. The other sessions allow program A or program B to communicate with other programs. During the communication between two programs, one program may send a message over the session to another program, requesting access to a local resource of the other program. In this way, a local resource of program B, for example, may become a remote resource of program A.

All communication provided by APPC is program-to-program and accommodates a variety of distributed processing connections. For example, a distributed parts inventory could make good use of APPC services. Each warehouse belonging to a company or division could maintain its individual physical inventory on an LU type 6.2 node. The corporate headquarters inventory could regularly compile inventory totals from all relevant warehouses. In order to locate a given part, a transaction programmer could write a program that inquired at each warehouse until the part was found. This process simplifies the task by reducing the number of inquiries that are sent to the host; instead, they are sent to each warehouse directly. When necessary, the host can be accessed for centralized information.

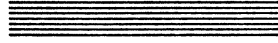








## **Chapter 2**



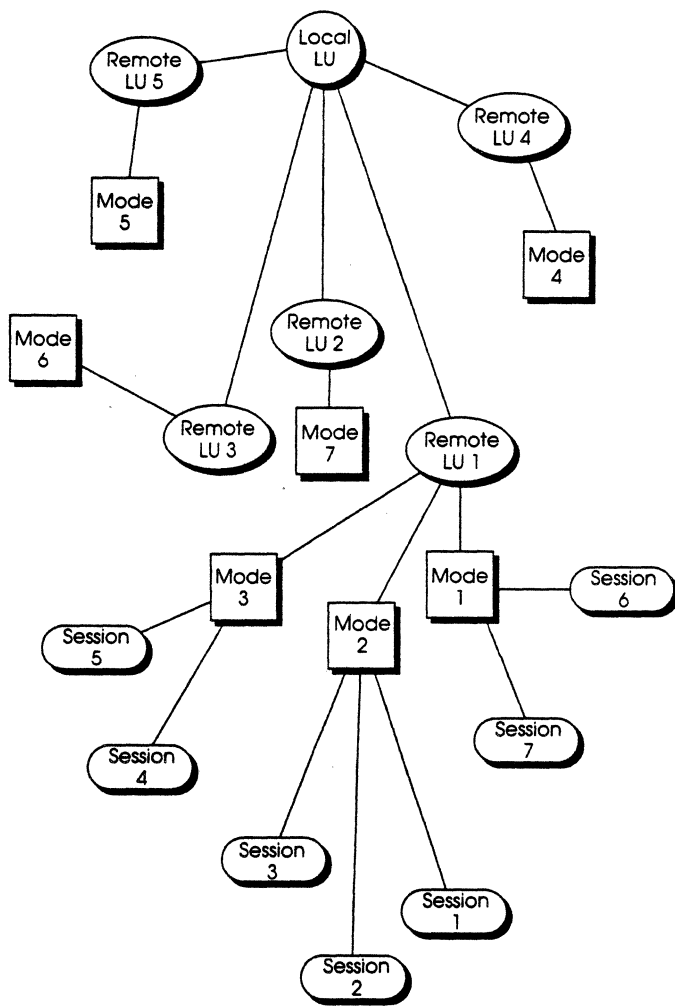
### **What Is MacAPPC?**



MacAPPC™ provides communications and administrative utilities that implement the elements of IBM's APPC architecture, including LU type 6.2 and PU type 2.1, on the Apple® Macintosh® computer. By conforming to IBM's LU 6.2 protocols, MacAPPC provides the basis for distributed transaction processing using Macintosh computers.

The task of building the components of a large network requires considerable time, effort, and expertise. The power designed into LU type 6.2 increases the complexity of this task by allowing for dynamic network configuration. The purpose of MacAPPC is to simplify the configuration process by providing sophisticated but easy-to-use tools that reduce complexity, speed data entry, and present options in a logical and usable format.

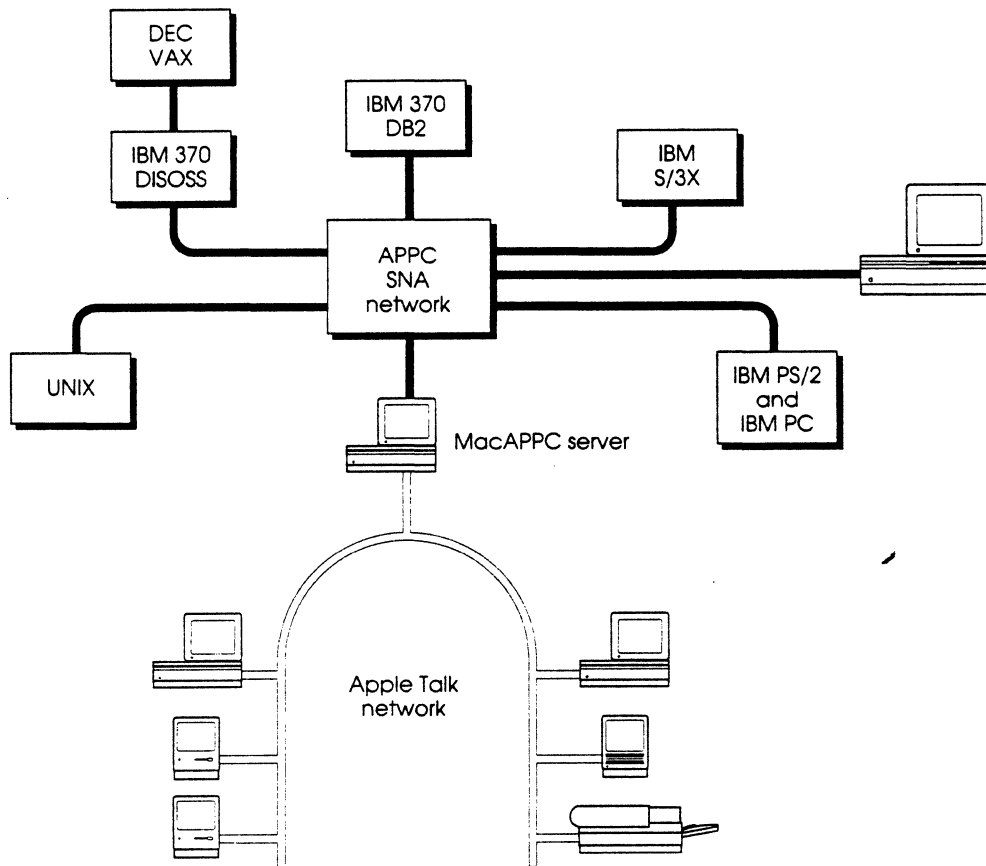
In a large network, the number of configuration options can appear almost infinite; for example, Figure 2-1 shows a representation of a hypothetical network configuration, showing resources such as local LUs, remote LUs, modes, and sessions. MacAPPC accommodates small and simple to extremely large and complex networks with equal facility.



**Figure 2-1**  
A hypothetical logical network structure



MacAPPC provides a programming interface to allow you to provide connectivity to any other system that uses APPC. It permits the development of a heterogeneous network of systems using Macintosh computers, IBM Personal Computers and compatibles, and mainframes and mini-computers of various models and vendors. With MacAPPC, you can integrate Macintosh computers with the communications network used by an estimated three-quarters of the major corporations in the United States, and by a large percentage elsewhere. In addition, because of its distributed nature, MacAPPC makes these capabilities available to each Macintosh computer on an AppleTalk® network system connected to a MacAPPC server. Figure 2-2 summarizes the scope of the MacAPPC environment.



**Figure 2-2**  
The MacAPPC environment



By making such connectivity possible, MacAPPC provides you (or your transaction programmers) with the means of writing transaction programs for the Macintosh computer in a distributed transaction environment. By virtue of the programming interface provided by MacAPPC, the Macintosh programmer has access to all the facilities of an SNA network.

MacAPPC includes a set of routines that conform to SNA's LU 6.2 design standards. These routines, which are implemented in MacAPPC as Macintosh drivers, provide the programming interface that allows a transaction programmer to use APPC functions on the Macintosh computer. The categories of MacAPPC routines are as follows:

- Conversation routines handle the exchange of data between TPs over basic or mapped conversations. These functions are accessed via the MacAPPC Conversation Driver.
- Control operator routines are used in controlling aspects of LU components. These functions are accessed via the MacAPPC Control Operator Driver.
- Node operator routines control aspects of PU components. These functions are accessed via the MacAPPC Node Operator Driver.
- Transaction program routines perform connection and utility functions. These functions are accessed via the MacAPPC Transaction Program Driver.

These routines are explained in detail in Part II, "MacAPPC Programmer's Reference." Later releases of MacAPPC will include additional functionality that will build on that provided by the current release of MacAPPC.

In addition, MacAPPC includes applications and utilities that allow for installation and operation of network components, such as logical units, physical units, sessions, and modes. The following applications and their respective functions are provided:

- The Administration program and Configuration program, which provide the means to define resources such as logical units and physical units, establish sessions, establish modes, and perform numerous other tasks.
- The MacAPPC server, which resides on an intelligent communications card in an expansion slot of a Macintosh II. The MacAPPC server provides full support for LU 6.2/PU 2.1 functions. It also provides network services for other Macintosh computers on an AppleTalk network system.
- The MacAPPC Chooser device, which permits the user to select any server on an AppleTalk network.

The functions of these components are further explained in Part III, "MacAPPC User's Guide."



---

---

## Macintosh user interface

MacAPPC provides all of the applications and utilities needed to configure, install, and activate your network. The Administration program, the Configuration program, and the MacAPPC Chooser device are the components that provide the Macintosh-defined user interface.

The usefulness of MacAPPC extends beyond the ability to write transaction programs. Equally important is the fact that your programming tasks and system access are made considerably easier by the Macintosh user interface. Through the use of icons, windows, scroll bars, dialog boxes, and other Macintosh user features, the MacAPPC administrative utilities provide clear and simple formats for building sessions; establishing addresses; defining logical units, physical units, and modes (including parameters); and performing the variety of other tasks that are associated with APPC.

For the MIS programmer, the interface provided by the Macintosh computer offers ease of use, rapid modification of a network configuration, and a high degree of visibility for such system features as the logging facility.

For the Macintosh software developer, MacAPPC provides easy connectivity to any computer that uses APPC. The Macintosh look and feel are the same as in other Macintosh software, and the Macintosh interface is maintained. In addition, the full range of Macintosh programming languages and programming flexibility is available.

---

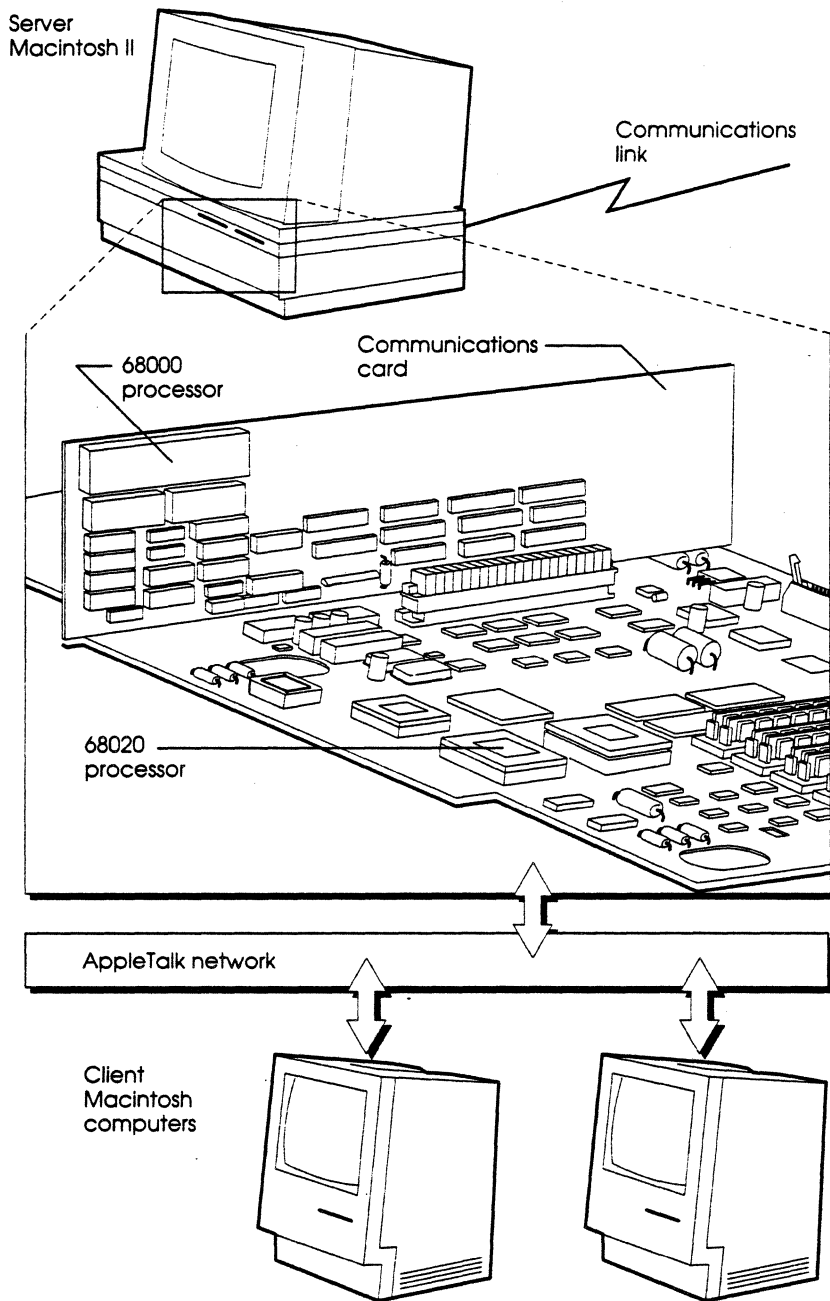
---

## Server-client architecture

The MacAPPC protocols are implemented by the MacAPPC server, a software utility that operates on an intelligent communications card residing in a slot on a Macintosh II computer. This card provides a server-client architecture that extends MacAPPC to other Macintosh computers—the clients—over an AppleTalk network system.

Because each MacAPPC server runs on a communications card, not on the Macintosh II, the computer is not dedicated to the server function. Any client on an AppleTalk network may access the MacAPPC server, permitting a programmer on that client to write and run transaction programs that use the MacAPPC routines. In fact, any computer that has access to the AppleTalk network system can function as a client of the MacAPPC server. See Figure 2-3 for an illustration of the server-client relationship.





**Figure 2-3**  
The MacAPPC server-client relationship



---

---

## Security

MacAPPC supports two levels of security: session level and conversation level.

Session level LU-LU verification is used to verify the identity of each LU to its session partner LU during activation of a session. A session between two LUs cannot be activated unless each LU's view of the other LU is the same. In essence, each node must see a mirror image of the other node, both in configuration and in security, in order for a session to be successfully bound.

Conversation-level access security information is carried on allocation requests in order for the receiving LU to verify the identity of the user ID, and to control access to its resources. The security information includes a user ID together with a password or the already-verified indication. The information may also include a **profile**, which is used at the conversation level to provide an additional element of structure and security to the network configuration. Some examples of profiles are department, store, corporation, section, division, function, building, location, and code designation.

For more information about security, see Chapter 5.

---

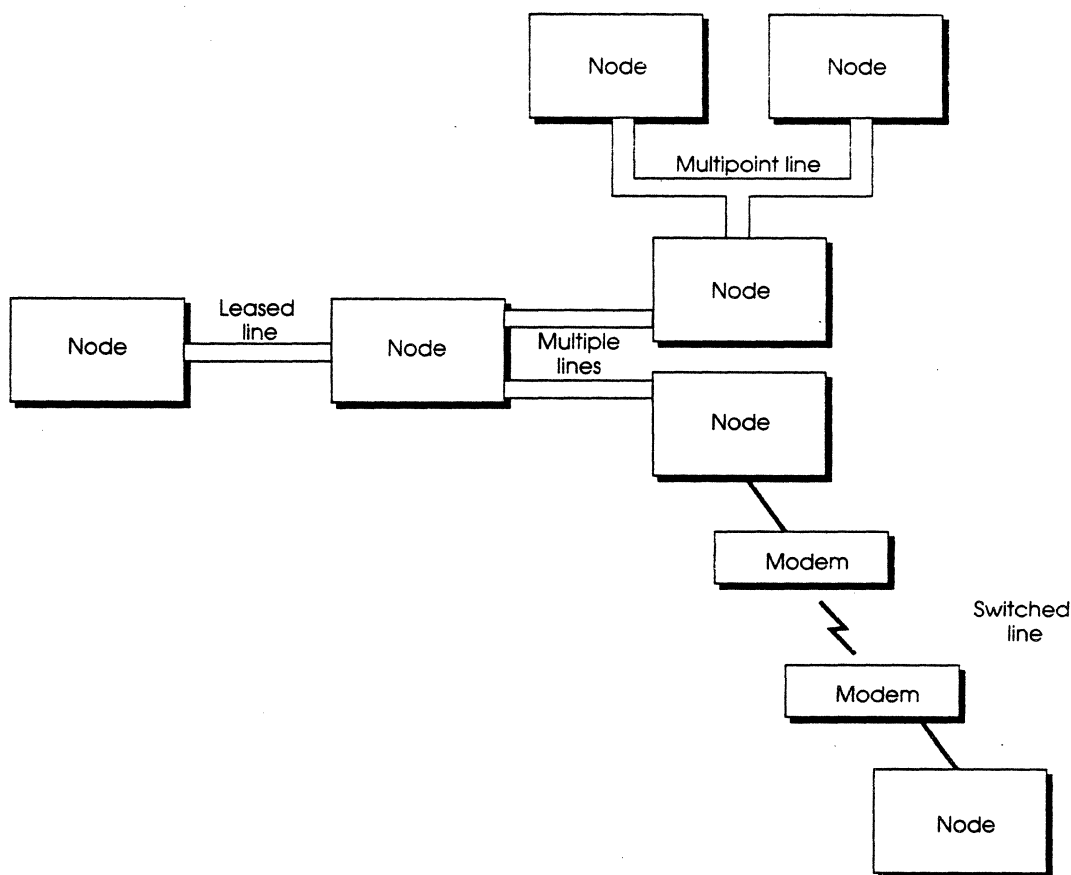
---

## Transmission media

MacAPPC is device independent, operating-system independent, and medium independent. At this time, IBM has implemented APPC on Synchronous Data Link Control (SDLC) and Token Ring. Currently, MacAPPC supports link connections that use IBM's SDLC protocols. As IBM extends the transmission capability of LU 6.2, MacAPPC can be expanded to accommodate those media as well.

MacAPPC supports leased, switched, and multipoint connections, as well as multiple links of these three connection types. A leased connection is a directly connected line. A switched connection (for example, a telephone line) is established when required and broken when a session is completed. A multipoint connection is a party line in which several users share the same line. Multiple links are multiple connections from one LU to other LUs. These connection types are shown schematically in Figure 2-4.





**Figure 2-4**  
MacAPPC connection types

---

## MacAPPC drivers

MacAPPC also provides a programming interface for transaction programs to a set of RAM device drivers. Macintosh programs call the MacAPPC routines that perform APPC functions by the use of drivers.

---

### MacAPPC Conversation Driver

LU 6.2 defines three types of conversation routines:

- Mapped conversation routines are used by application programs at a high level to exchange data records.
- Basic conversation routines are a low-level interface used by service TPs or by other specialized application TPs.
- Type-independent conversation routines are used with both mapped and basic routines to perform certain functions, such as wait for posting.



When mapped routines are used, the data does not need to be formatted by the application program. Basic routines are normally used only by service programs—the specially defined programs which provide common services across the network, such as the change-number-of-sessions (CNOS) routines.

The difference between mapped and basic conversations is significant. In mapped conversations, the data must be formatted into **General Data Stream (GDS)** variables. GDS variables consist of two bytes containing the length of the variable, two bytes containing an IBM-assigned code identifying the type of variable, and then data. In basic conversations the data has less internal formatting—only the length and the data are included. Mapped routines handle all of the formatting for mapped conversations (making it transparent to the application). Basic routines handle none of the formatting; instead, data must be presented to the routine packed in the correct format. For this reason, mapped routines are usually considered easier to use.

---

## MacAPPC Control Operator Driver

SNA specifies a control operator function for an LU, but does not define it; that is, the control operator may be an individual or a program. Control operator routines define the protocol boundary for a control operator transaction program, which performs control operator functions for the LU. Control operator routines are divided into the following categories:

- ☐ Change-number-of-sessions (CNOS) routines change, display, initialize, and reset the number of sessions for a given mode.
- ☐ Session-control routines activate and deactivate an LU–LU session.
- ☐ LU definition routines define, modify, display and examine operating parameters for a local LU, remote LU, and mode.

The LU control operator describes and controls the availability of certain resources: for example, it describes network resources accessed by the local LU, and it controls the number of sessions between the LU and its partners.

---

## MacAPPC Node Operator Driver

Node operator routines are used to define and control the components of a PU 2.1 node. Their function is not specified by IBM; however, it is implied by the properties that are assigned to the node by SNA. Node operator routines are divided into the following categories:

- ☐ Node control routines are used to activate and deactivate node components.
- ☐ Node message routines are used to define and display node message queues and messages.
- ☐ Node definition routines are used to define, display, and delete node components.

---

## MacAPPC Transaction Program Driver

Transaction program routines are used to perform various transaction program tasks. The connection between a transaction program and a MacAPPC server is known as an **attach**. Each attach creates a separate logical instance of a program.



Transaction program routines are divided into the following categories:

- ☐ Transaction program connection routines are used to attach and detach to a server or an LU.
- ☐ Transaction program utility routines convert application-specific ASCII data to EBCDIC, and vice versa.

A transaction program must attach to a server or LU before it can issue any conversation, control operator, or node operator routines. Multiple attach requests may be issued so the program can issue routines to more than one MacAPPC server or LU. Each of these attaches creates a new logical instance of the program.

---

---

## MacAPPC server

The MacAPPC server provides full support for LU 6.2 and PU 2.1 functions. The MacAPPC server is data-link independent. It resides on an intelligent communications card in an expansion slot of a Macintosh II computer. The operating system of the card is independent of the Macintosh II operating system.

The MacAPPC server is also capable of providing network services for other Macintosh computers on an AppleTalk network, in a server-client relationship. Each server (and its clients) is seen as a single node to the SNA network.

The Macintosh II is not dedicated to the MacAPPC server (the communications card performs this role); rather, it is independent of the server, and in fact may function as a client of the server.

The MacAPPC server includes the following functions:

- ☐ PU 2.1 functions, including multiple sessions (up to 254), parallel sessions (up to 254), and PU 2.0 support (peripheral node).
- ☐ AppleTalk request and reply transactions, which provide transport of APPC messages between the MacAPPC server and transaction programs on the clients.
- ☐ Two levels of security: conversation level and session level.

The SDLC functions include primary, secondary, and negotiable roles, multipoint line support, multiple physical-link support, and leased and switched-line support.

---

---

## MacAPPC Chooser device

The MacAPPC Chooser device is used to select a particular MacAPPC server from those that you have loaded onto a communications card on your network. The Chooser device displays the current selection, which can be either

- ☐ internode (a MacAPPC server residing in a Macintosh II accessed across AppleTalk network system), or
- ☐ intranode (a MacAPPC server residing in the same Macintosh II as the MacAPPC Chooser device).



The human interface takes the form of a MacAPPC server icon, along with a list of active MacAPPC servers in the currently selected zone. You select a MacAPPC server by clicking the appropriate server icon. See Chapter 10, "Selecting a MacAPPC Server," for additional information.

---

---

## **MacAPPC Configuration program**

The purpose of the Configuration program is to create a configuration file that describes the components of a MacAPPC network. Components can be created, edited, and deleted. In addition, you can describe conversation-level security parameters for local LUs and TPs and session-level security parameters for local and remote LUs. This file can then be read by the Administration program and used to configure the server. Thus, the Configuration program does not use MacAPPC driver routines, but rather builds a description of the network, which the Administration program can then implement. See Chapter 11, "The MacAPPC Configuration Program," for additional information.

---

---

## **MacAPPC Administration program**

The purpose of the Administration program is to provide you with a ready-made way to load the MacAPPC server onto the communications card, create a network configuration using a configuration file created by the Configuration program, and manage these configured network components. The Administration program uses MacAPPC control operator and node operator routines to define, display, and manage the network.

The Administration program does not dynamically modify the MacAPPC server; therefore, each time you change a component setting in the configuration file, the MacAPPC server must be restarted. See Chapter 12, "The MacAPPC Administration Program," for additional information.

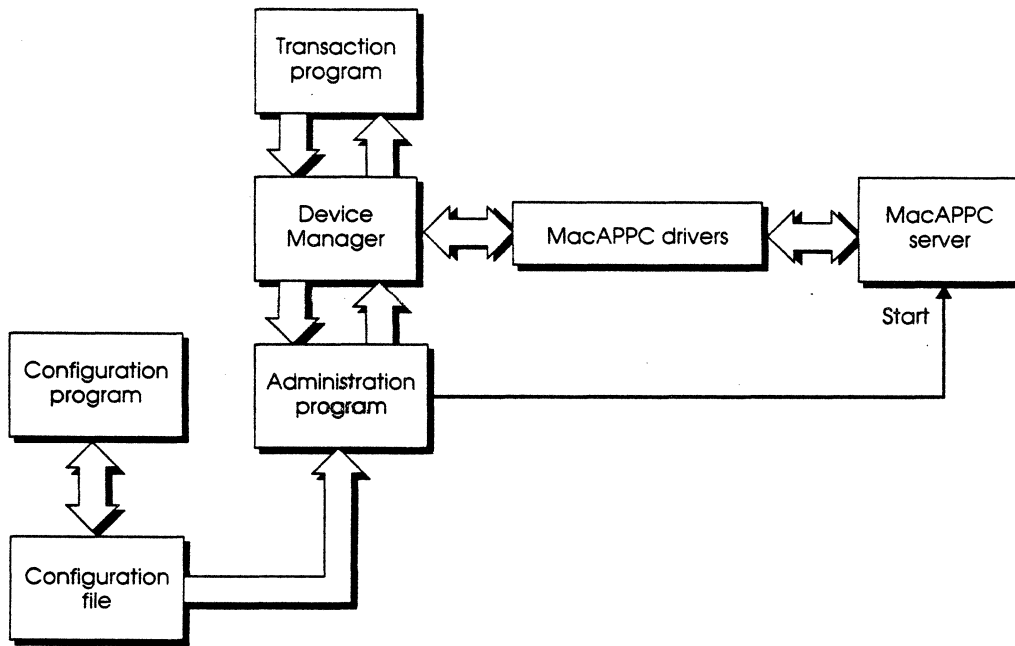
---

---

## **MacAPPC software and server relationship**

As you plan your use of MacAPPC, it is important to keep in mind the functional relationships between the MacAPPC software and the server. Figure 2-5 schematically illustrates the interrelationships between the Administration program, the Configuration program, the MacAPPC server, and the drivers that are used to call the MacAPPC routines.

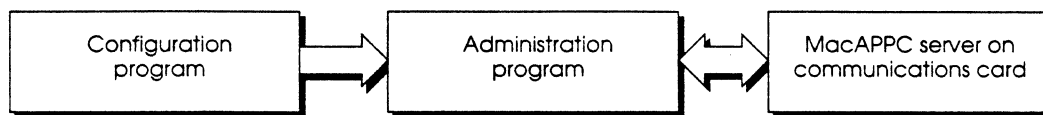




**Figure 2-5**  
MacAPPC interactions

Figure 2-6 summarizes the functions and interactions of the Configuration program, the Administration program, and the server.





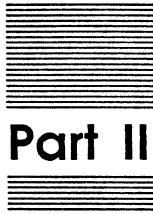
- |  |   |  |
|--|---|--|
| <ul style="list-style-type: none"> <li><input type="checkbox"/> Provides user interface for definition and modification of network component parameters</li> <li><input type="checkbox"/> Provides user interface for definition and modification of network security parameters</li> <li><input type="checkbox"/> Provides logic error checking of formats and configuration parameters</li> <li><input type="checkbox"/> Generates configuration file describing the physical and logical components of the network</li> </ul> | <ul style="list-style-type: none"> <li><input type="checkbox"/> Reads configuration resource file</li> <li><input type="checkbox"/> Downloads the server to the communications card</li> <li><input type="checkbox"/> Uses MacAPPC control operators and node operators to define and manage the network</li> <li><input type="checkbox"/> Starts and stops server</li> <li><input type="checkbox"/> Activates and deactivates LUs, lines, stations, and sessions</li> <li><input type="checkbox"/> Initializes and resets session limits</li> <li><input type="checkbox"/> Provides real-time network logging</li> </ul> | <ul style="list-style-type: none"> <li><input type="checkbox"/> Provides program-to-program communication between transaction programs</li> <li><input type="checkbox"/> Provides interface link to network</li> </ul> |
|--|---|--|

**Figure 2-6**  
MacAPPC programs and server relationship









## **Part II**

# **MacAPPC Programmer's Reference**

This part consists of six chapters that provide a detailed programming reference for MacAPPC, Apple's implementation of APPC on the Apple Macintosh computer.

Chapter 3 describes the general structure of the MacAPPC drivers and explains how to construct a Macintosh program using the drivers.

Chapter 4 provides a detailed description of each MacAPPC conversation driver routine and includes a summary of the conversation driver constants, data structures, and routines at the end of the chapter.

Chapter 5 provides a detailed description of each MacAPPC control operator routine and includes a summary of the control operator constants, data structures, and routines at the end of the chapter.

Chapter 6 provides a detailed description of each MacAPPC node operator routine and includes a summary of the node operator constants, data structures, and routines at the end of the chapter.


Chapter 7 provides a detailed description of each MacAPPC transaction program routine and includes a summary of the transaction program constants, data structures, and routines at the end of the chapter.

Chapter 8 shows an example of a fragment of a transaction program.

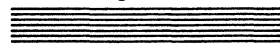








## **Chapter 3**



### **MacAPPC Drivers**



Part of MacAPPC has been implemented as a set of Macintosh device drivers. The MacAPPC drivers provide the programmatic interface for transaction programs to perform APPC functions.

This chapter describes the MacAPPC drivers in detail. You should already be familiar with

- the Memory Manager, including the allocation of memory in the heap and the use of pointers and handles, as discussed in Chapter 1 of Volume II of *Inside Macintosh*
- the Device Manager, including the handling of interrupts and the use of devices and device drivers, as described in Chapter 6 of Volume II of *Inside Macintosh*

---

---

## Using MacAPPC drivers

A device driver can be either opened or closed. After a driver has been opened, an application can transmit control information to it, read data from it, and write data to it. Before it is opened, you identify a driver by its driver name; after it is opened, you identify it by its reference number.

You access the MacAPPC drivers by making Macintosh Device Manager calls. Chapter 6, "The Device Manager," in *Inside Macintosh*, Volume II will be an important resource for you as you learn how to use the MacAPPC drivers.

The MacAPPC drivers are:

- the MacAPPC Conversation Driver, .CV62
- the MacAPPC Control Operator Driver, .CO62
- the MacAPPC Node Operator Driver, .NO62
- the MacAPPC Transaction Program Driver, .TP62

To begin using the MacAPPC routines, you must open the .CV62, .CO62, .NO62, and .TP62 drivers by using the PBOpen or OpenDriver routine. You can then make control calls to any of the MacAPPC drivers requesting a particular MacAPPC routine to be performed. The interface files specify the data structures and constants you must use when making control calls to the MacAPPC drivers. Chapters 4 through 7 document this interface, and Appendixes A and B list the interface and error files.

Each driver contains a set of routines, as shown in Table 3-1.



**Table 3-1**  
MacAPPC drivers and the categories of routines

<b>.CV62</b>	<b>.CO62</b>	<b>.NO62</b>	<b>.TP62</b>
Conversation routines	Control operator routines	Node operator routine	TP routines
<b>Mapped routines</b>	<b>CNOS routines</b>	<b>Node control routines</b>	<b>Connection routines</b>
MCAAllocate	COChangeSessionLimit	NOActivateLine	TPAttach
MCConfirm	COInitializeSessionLimit	NOActivateLU	TPDetach
MCConfirmed	COProcessSessionLimit	NOActivateNode	<b>Utility routines</b>
MCDeallocate	COResetSessionLimit	NOActivateStation	TPAsciiToEbcDic
MCFlush	<b>Session control routines</b>	NODEactivateLine	TPEbcDicToAscii
MCGetAttributes	COActivateSession	NODEactivateLU	
MCPPostOnReceipt	CODEactivateSession	NODEactivateNode	
MCPPrepareToReceive	<b>LU definition routines</b>	NODEactivateStation	
MCReceiveAndWait	CODefineLocalLU	<b>Node message routines</b>	
MCReceiveImmediate	CODefineMode	NoDefineMessageQueue	
MCRequestToSend	CODefineRemoteLU	NODisplayMessage	
MCSendData	CODefineTP	NoDisplayMessageQueue	
MCSendError	CODElete	<b>Node definition routines</b>	
MCTest	COisplayLocalLU	NODEfineCP	
<b>Type-independent routines</b>	CODisplayMode	NODEfineLine	
CVBackout	CODisplayRemoteLU	NODEfineNode	
CVGetType	CODisplaySession	NODEfineStation	
CVSyncPoint	CODisplayTP	NODElete	
CVWait		NODisplayCP	
<b>Basic routines</b>		NODisplayLine	
BCAllocate		NODisplayNode	
BCConfirm		NODisplayStation	
BCConfirmed			
BCDeallocate			
BCFlush			
BCGetAttributes			
BCPostOnReceipt			
BCPrepareToReceive			
BCReceiveAndWait			
BCReceiveImmediate			
BCRequestToSend			
BCSendData			
BCSendError			
BCTest			



---

---

## Synchronous and asynchronous execution

You can execute MacAPPC routines either synchronously or asynchronously.

When your application executes a **synchronous** MacAPPC routine, the application will not continue until the MacAPPC routine is completed.

When your application executes an **asynchronous** MacAPPC routine, an I/O request is placed in the appropriate driver's I/O queue, and control returns immediately to the executing program—possibly even before the actual I/O is completed. Requests are taken from the queue one at a time, and processed; meanwhile, the executing program is free to work on other things.

Routines that are executed asynchronously return control to the executing program with the result code `noErr` as soon as the routine is placed in the driver's queue. This doesn't indicate that the routine completed successfully; it simply indicates that the routine was successfully queued to the appropriate driver. To determine when the routine is actually completed, poll the `ioResult` field of the routine's parameter block. The `ioResult` field, set to `appcExec` when the routine is executed, receives the actual result code when the routine is completed.

Your application may specify a completion routine to be executed at the end of an asynchronous operation by setting the `ioCompletion` field to a pointer to that routine.

❖ *Note:* See the Device Manager chapter in *Inside Macintosh* for information and restrictions on writing completion routines.

---

---

## MacAPPC driver parameter blocks

Routine parameters passed by an application to the Device Manager and routine parameters returned by the Device Manager to an application are contained in a parameter block, which is a data structure allocated by the application.

You pass a different type of parameter block for each of the MacAPPC drivers, as shown in Table 3-2.

**Table 3-2**  
MacAPPC drivers and their parameter blocks

Driver	Parameter block
.CV62	cvParam
.CO62	coParam
.NO62	noParam
.TP62	tpParam



The definition for each of those parameter blocks is contained in the variable-length data structure `APPCParamBlock`, as follows:

```
APPCParamType =      (cvParam, coParam, noParam, tpParam);

APPCParamBlock = RECORD
    qLink:           QElemPtr;    {DRVR QElem pointer}
    qType:           INTEGER;      {DRVR queue type}
    ioTrap:          INTEGER;      {DRVR IO trap}
    ioCmdAddr:       Ptr;          {DRVR IO command pointer}
    ioCompletion:    ProcPtr;      {DRVR IO completion routine pointer}
    ioResult:        OSErr;        {DRVR IO result}
    ioNamePtr:       StringPtr;    {DRVR IO name pointer}
    ioVRefNum:       INTEGER;      {DRVR IO Volume refNum}
    appcRefNum:      INTEGER;      {APPC driver refNum}
    appcOpCode:      INTEGER;      {APPC type of call}
    appcHiResult:    INTEGER;      {APPC major result code}
    appcLoResult:    INTEGER;      {APPC minor result code}
    appcConvState:   Byte;         {APPC conversation state}
    appcUserRef:     LONGINT;      {for your use}
    CASE APPCParamType OF
        cvParam:
            . . . {CV parameters}
        coParam:
            . . . {CO parameters}
        noParam:
            . . . {NO parameters}
        tpParam:
            . . . {TP parameters}
    END;
END;

APPCParamBlockPtr =      ^APPCParamBlock;
APPCParamBlockHandle=    ^APPCParamBlockPtr;
```

The particular fields for each type of parameter block are given in the summary at the end of the corresponding chapter.

The maximum size of an `APPCParamBlock` data structure in bytes is specified by one of the following constants:

```
kAPPCSize =      2706;    { Maximum APPC parameter block size      }
kCVSize =        2706;    { Conversation parameter block size    }
kCOSize =        192;    { Control Operator parameter block size }
kNOSize =        108;    { Node Operator parameter block size  }
kTPSize =        1628;    { Transaction Program parameter block size }
```

---

## qLink, qType, and the I/O fields

The first four fields in each parameter block are handled entirely by the Device Manager, and you usually don't need to be concerned with them. Those fields, along with the four additional I/O fields, are documented in the Device Manager chapter in *Inside Macintosh*.



Your application can use the `ioCompletion` field to specify a completion routine that will be executed when an asynchronous request is completed.

❖ *Note:* See the Device Manager chapter in *Inside Macintosh* for information and restrictions on writing completion routines.

The `ioResult` field is used to indicate the completion status of the MacAPPC routine in progress.

The `ioNamePtr` and `ioVRefNum` fields are not used by MacAPPC drivers.

---

## appcRefNum

The `appcRefNum` field must contain the driver reference number returned when the device driver is opened. You must supply the appropriate value for every MacAPPC routine.

---

## appcOpCode

The `appcOpCode` field specifies the MacAPPC routine to be executed. Use the name of the routine preceded by a lowercase letter *k*, as follows:

<code>kMCAAllocate =</code>	<code>100;</code>	<code>{ Mapped conversation routines }</code>
<code>kMCConfirm =</code>	<code>101;</code>	
<code>kMCConfirmed =</code>	<code>102;</code>	
<code>kMCDeallocate =</code>	<code>103;</code>	
<code>kMCFlush =</code>	<code>104;</code>	
<code>kMCGetAttributes =</code>	<code>105;</code>	
<code>kMCPostOnReceipt =</code>	<code>106;</code>	
<code>kMCPrepareToReceive =</code>	<code>107;</code>	
<code>MCReceiveAndWait =</code>	<code>108;</code>	
<code>kMCReceiveImmediate =</code>	<code>109;</code>	
<code>kMCRequestToSend =</code>	<code>110;</code>	
<code>kMCSendData =</code>	<code>111;</code>	
<code>kMCSendError =</code>	<code>112;</code>	
<code>kMCTest =</code>	<code>113;</code>	
<code>kCVBackout =</code>	<code>114;</code>	<code>{ Type independent routines }</code>
<code>kCVGetType =</code>	<code>115;</code>	
<code>kCVSyncPoint =</code>	<code>116;</code>	
<code>kCVWait =</code>	<code>117;</code>	
<code>kBCAllocate =</code>	<code>118;</code>	<code>{ Basic conversation routines }</code>
<code>kBCConfirm =</code>	<code>119;</code>	
<code>kBCConfirmed =</code>	<code>120;</code>	
<code>kBCDeallocate =</code>	<code>121;</code>	
<code>kBCFlush =</code>	<code>122;</code>	
<code>kBCGetAttributes =</code>	<code>123;</code>	
<code>kBCPostOnReceipt =</code>	<code>124;</code>	
<code>kBCPrepareToReceive =</code>	<code>125;</code>	
<code>kBCReceiveAndWait =</code>	<code>126;</code>	
<code>kBCReceiveImmediate =</code>	<code>127;</code>	
<code>kBCRequestToSend =</code>	<code>128;</code>	



kBCSendData =	129;	
kBCSendError =	130;	
kBCTest =	131;	
kCOChangeSessionLimit =	200;	{ Control operator CNOS routines }
kCOInitializeSessionLimit =	202;	
kCOProcessSessionLimit =	203;	
kCOResetSessionLimit =	204;	
kCOActivateSession =	205;	{ Control operator session control routines }
kCODEactivateSession =	206;	
kCODEfineLocalLU =	207;	{ Control operator LU definition routines }
kCODEfineRemoteLU =	208;	
kCODEfineMode =	209;	
kCODEfineTP =	210;	
kCODisplayLocalLU =	211;	
kCODisplayRemoteLU =	212;	
kCODisplayMode =	213;	
kCODisplaySession =	214;	
kCODisplayTP =	215;	
kCODElete =	216;	
kNOActivateLine =	300;	{ Node operator node control routines }
kNOActivateLU =	301;	
kNOActivateNode =	302;	
kNOActivateStation =	303;	
kNODEactivateLine =	305;	
kNODEactivateLU =	306;	
kNODEactivateNode =	307;	
kNODEactivateStation =	308;	
kNODEfineMessageQueue =	304;	{ Node operator node message routines }
kNODisplayMessage =	310;	
kNODisplayMessageQueue =	309;	
kNODEfineNode =	311;	{ Node operator node definition routines }
kNODEfineCP =	312;	
kNODEfineLine =	313;	
kNODEfineStation =	314;	
kNODisplayNode =	316;	
kNODisplayCP =	317;	
kNODisplayLine =	318;	
kNODisplayStation =	319;	
kNODElete =	320;	
kTPAttach =	400;	{ Transaction program connection routines }
kTPDetach =	401;	
kTPAsciiToEbcdic =	403;	{ Transaction program utility routines }
kTPEbcdicToAscii =	404;	

Each routine is described in detail in Chapters 4 through 7 in this guide.



---

## appcHiResult and appcLoResult

The values of the `appcHiResult` and `appcLoResult` fields are inserted by MacAPPC when the routine is completed. See Appendixes B and C for the meanings of the result codes.

---

## appcConvState

The value of the `appcConvState` field is inserted by MacAPPC when the routine is completed, and is always a member of the following set of constants:

<code>kNullState =</code>	<code>0;</code>	
<code>kResetState =</code>	<code>1;</code>	
<code>kSendState =</code>	<code>2;</code>	
<code>kReceiveState =</code>	<code>3;</code>	
<code>kConfirmState =</code>	<code>4;</code>	
<code>kConfirmSendState =</code>	<code>5;</code>	
<code>kConfirmDeallocState =</code>	<code>6;</code>	
<code>kDeallocState =</code>	<code>7;</code>	
<code>kDeferState =</code>	<code>8;</code>	<code>{ not supported }</code>
<code>kSyncPtState =</code>	<code>9;</code>	<code>{ not supported }</code>
<code>kBackedOutState =</code>	<code>10;</code>	<code>{ not supported }</code>

---

## appcUserRef

Your application can use the `appcUserRef` field in any way it wants; the MacAPPC drivers do not use the field.

---

---

## MacAPPC driver control blocks

A transaction program is responsible for allocating and maintaining memory that the MacAPPC device drivers use for their own purposes. This section defines these special types of memory blocks.

---

### Important

All of the blocks defined in this section are the exclusive property of the device driver; they must remain locked and may not be altered or moved while being used.

---

---

## Transaction Program Control Block (TPCB)

Your transaction program may need to allocate and maintain a block of memory called the **Transaction Program Control Block (TPCB)**. The MacAPPC device drivers use this block to maintain state and control information about an individual connection to a MacAPPC server.



Once the routine that initiates the connection succeeds, the TPCB becomes the exclusive property of the device driver and must remain locked and not be altered or moved until the connection with the MacAPPC server is terminated. When the connection is terminated, your application may deallocate the TPCB.

Certain MacAPPC routines require you to supply a pointer to a new TPCB when a connection is initiated. The same TPCB pointer must be supplied to every routine executed for a particular connection once that connection has been established.

---

## Conversation Control Block (CVCB)

Your transaction program may also need to allocate and maintain a block of memory called the **Conversation Control Block (CVCB)**. The MacAPPC device drivers use this block to maintain state and control information about an individual conversation with a partner transaction program.

Once the routine that initiates the conversation succeeds, the CVCB becomes the exclusive property of the device driver. The block must remain locked and not be altered or moved until the conversation with the partner TP is terminated. When the conversation is terminated, your application may deallocate the CVCB.

Certain MacAPPC routines require you to supply a pointer to a new CVCB when a conversation is initiated. The same CVCB pointer must be supplied to every routine executed for a particular conversation once that conversation has been established.

---

## PIP buffer

If the transaction program is using PIP data, it must also allocate and maintain a block of memory called the **PIP buffer**. The MacAPPC device drivers use this block to hold any PIP (program initialization parameter) data that may be sent or received from a partner transaction program.

The PIP buffer is the exclusive property of the device driver. The block must not be altered or moved for the duration of the conversation. This buffer must be large enough to hold the largest amount of PIP data expected plus a 4-byte logical length ID (LLID) per parameter plus one 4-byte LLID for the entire PIP data.

---

## Mapped conversation buffer

If the transaction program is participating in a mapped conversation, it must also allocate and maintain a block of memory called the **mapped conversation buffer**. The MacAPPC device drivers use this block to hold any data sent or received from a partner transaction program over a mapped conversation before the actual mapping of the data is performed.

The mapped conversation buffer is the exclusive property of the device drivers. The block must not be altered or moved for the duration of the mapped conversation. This buffer must be large enough to hold the largest complete data record expected plus a 4-byte logical length ID (LLID). The buffer may be any size needed by the transaction program and may be tuned for performance enhancements.



---

---

## MacAPPC driver constants

The following constants are used in all MacAPPC drivers:

### { Block Sizes }

```
kTPCBlockSize =      3000;    { Transaction Program Control Block size }
kCVCBSize =         3500;    { Conversation Control Block size }

kLineSize =          17;      { Maximum Line size }
kSDLCSize =          17;      { SDLC Line size }

kMsgSize =           44;      { Message Structure size }
kMsgFieldSize =      2;       { Message Data Field size }

kMaxPIP =            256;     { Maximum # of PIPs }
kMaxCVCB =            256;     { Maximum # of CVCB pointers }
```

### { String Sizes }

```
kMaxName =           8;       { Maximum generic string length }
kMaxTPName =          64;      { Maximum TP Name length }
kMaxSecName =         10;      { Maximum Security Field length }
kMaxMapName =          64;      { Maximum Map Name length }
kMaxPhoneNumber =     20;       { Maximum phone number length }
kMaxLogData =         200;     { Maximum log data length }
kMaxLUWName =         17;      { Maximum LUW name length }
kMaxLUWID =           6;       { Maximum LUW ID length }
kMaxLUWCorr =         8;       { Maximum LUW correlator length }
kMaxLUPswd =          16;      { Maximum LU-LU password length (hex) }
kMaxExchID =           8;      { Maximum Exchange ID length (hex) }
kMaxCPUID =           12;      { Maximum CPU ID length (hex) }
kMaxSDLCAddr =        2;       { Maximum SDLC address length (hex) }
```

### { Define Operation Values }

```
kIgnoreParam =       (-1);
kFuncNotSupp =        0;
kFuncSupp =           1;
kReplaceParam =       0;
kDeleteParam =        1;
kAddParam =           0;
kNextEntry =          0;
```



---

---

## MacAPPC driver IDs

The MacAPPC server allocates and maintains several IDs that your program can use to identify an attached transaction program, a conversation, or a session. These IDs are returned by certain MacAPPC routines.

---

### Program ID

The program ID identifies a particular attached transaction program; it is a unique number generated by the MacAPPC server.

---

### Conversation ID

The conversation ID identifies an individual conversation in use by a pair of transaction programs; it is a unique number generated by the MacAPPC server. A conversation in use by a pair of transaction programs is represented by two unique conversation IDs, one from each transaction program's perspective.

---

### Session ID

The session ID identifies an individual session in use by a pair of logical units over a particular mode; it is a unique number generated by the MacAPPC server. A session in use by a pair of modes is represented by two unique session IDs, one from each mode's perspective.

---

---

## Executing a MacAPPC driver routine

In summary, here are the steps necessary to execute a single MacAPPC routine:

1. Set `appcRefNum` to the number returned by the `PBOpen` call.
2. Set `appcOpCode` to the constant equivalent to the appropriate MacAPPC routine (for example `kTPAttach` for the `TPAttach` routine).
3. Set `ioCompletion` and `appcUserRef` if desired.
4. Supply any necessary additional values for the parameters for the MacAPPC routine, as detailed in Chapters 4 through 7 of this guide.
5. Call `PBControl`, supply a pointer to the MacAPPC routine's parameter block, and set the `ASYNC` parameter in the `PBControl` call to `TRUE` or `FALSE`.
6. If you executed an asynchronous routine and did not supply your own `ioCompletion` routine, poll `ioResult` in the MacAPPC routine's parameter block to determine when the MacAPPC routine is completed, as detailed in "Synchronous and Asynchronous Execution," earlier in this chapter.
7. If `ioResult` indicates that an error occurred, examine `appcHiResult` and `appcLoResult` to determine the result code.



---

---

## MacAPPC driver conventions

In the following four chapters, each routine description includes a list of the `APPCParamBlock` fields affected by the routine. A 4-digit hex number gives the offset of the field within the parameter block, and its size is specified as byte, word, or long. When the size is not shown, the parameter is an array, as indicated by the convention of a pair of brackets (`[]`) following the parameter name.

The arrow next to each parameter name indicates the following:

Arrow	Meaning
→	Parameter is passed to the MacAPPC routine (described as <i>supplied</i> )
←	Parameter is returned by the MacAPPC routine (described as <i>returned</i> )
↔	Parameter is passed to and returned by the MacAPPC routine (described as <i>supplied/returned</i> )
⇒	Parameter points to space that is modified by the MacAPPC routine (described as <i>supplied/modified</i> )

In the parameter descriptions, the term **NIL pointer** is used to indicate that the pointer is set to a value of zero. Similarly, the term **NULL value** means a value of zero. The terms **TRUE** and **FALSE** are used as Boolean values; the particular values of TRUE or FALSE depends upon the language being used.





## **Chapter 4**



# **MacAPPC Conversation Driver**



This chapter describes the MacAPPC Conversation Driver (.CV62), explains how to use the driver, and provides a detailed guide to the programmatic interface for executing each Conversation Driver routine. For quick reference, a section at the end of the chapter summarizes the data structures, constants, and routine parameters.

---

---

## Using the MacAPPC Conversation Driver

The following sections document the Pascal-language interface to MacAPPC routines that provide support for Advanced Program-to-Program Communication (APPC) using either basic or mapped conversations. **Mapped conversations** use mapped conversation routines, which are more commonly used than basic conversation routines when programming in a high-level language. When mapped routines are used, the data does not need to be formatted by the application program. **Basic conversations** use basic conversation routines. Basic conversation routines are normally used only by service TPs, which are the programs that provide common services across the network, including overhead management, error handling, simultaneous activation, and security.

In mapped conversations, data is automatically formatted into **General Data Stream (GDS)** variables. GDS variables consist of 2 bytes containing the length of the variable, 2 bytes containing an IBM-assigned code identifying the type of variable, and then data. Mapped routines handle all of the formatting for mapped conversations (making it transparent to the application). In basic conversations the data has less internal formatting (only the length and the data are required). Basic routines handle none of the formatting—data must be presented to the routine packed in the correct format. Because mapped routines handle their own formatting, they are usually considered easier to use.

---

## Buffering

Each LU in a conversation has a buffer for sending and receiving data. When the transaction program executes a routine that sends data, it specifies an area containing the data, and the LU moves the data to its send buffer. The LU transmits the data, (**flushes** its send buffer) either when a sufficient amount of data is accumulated, or when the program executes a routine that explicitly causes the LU to transmit the accumulated data.

As incoming data arrives on a conversation, the LU places the data in its receive buffer. When the program executes a routine that receives data, it specifies an area in which the LU is to place the data. The LU moves the requested amount of data from the front of its receive buffer to the area specified by the program. In this way, the LU can accumulate incoming data in its receive buffer in advance of the program issuing the routine, or routines, that receive the data.

Routines that send data place the outgoing data in the send buffer behind any data from previous routines. Routines that send information other than data place the outgoing information in front of the information already in the buffer. A receiving LU accumulates incoming information in its receive buffer in the order in which it is received. The amount of buffered data that is sufficient for transmission is determined by the maximum size request/response unit (RU) that can be sent on the session.



## Conversation states

Conversations have states that determine what routines a transaction program can execute during the conversation. The following states are defined:

- ☐ reset
- ☐ send
- ☐ receive
- ☐ confirm
- ☐ confirm/send
- ☐ confirm/deallocate
- ☐ deallocate

The `appcConvState` constants are listed in Chapter 3, "The MacAPPC Drivers." Tables 4-1, 4-2, and 4-3 show the conversation routines and the states from which these routines can be executed.

**Table 4-1**  
States for mapped conversation routines

Mapped routine	Reset	Send	Receive	Confirm	Conf/send	Conf/dealc	Deallocate
MCAAllocate	X						
MCConfirm		X					
MCConfirmed				X	X	X	
MCDeallocate*							
flush		X					
confirm		X					
sync-level		X					
abend		X	X	X	X	X	
local							X
MCFlush		X					
MCGetAttributes		X	X	X	X	X	X
MCPPostOnReceipt			X				
MCPPrepareToReceive		X					
MCReceiveAndWait		X	X				
MCReceiveImmediate			X				
MCRequestToSend			X	X			
MCSendData		X					
MCSendError		X	X	X	X	X	
MCTest†							
posted			X				
request to send		X	X				

\* The state from which the routine can be executed varies depending on the type of deallocation; see "MCDeallocate," later in this chapter.

† The state from which the routine can be executed varies depending on the type of testing; see "MCTest," later in this chapter.



**Table 4-2**

States for type-independent conversation routines

Type-independent routine	Reset	Send	Receive	Confirm	Conf/send	Conf/dealc	Deallocate
CVGetType		X	X	X	X	X	X
CVWait			X				

**Table 4-3**

States for basic conversation routines

Basic routine	Reset	Send	Receive	Confirm	Conf/send	Conf/dealc	Deallocate
BCAllocate	X						
BCConfirm		X					
BCConfirmed				X	X	X	
BCDeallocate*							
flush		X					
confirm		X					
sync-level		X					
abend		X	X	X	X	X	
local							X
BCFlush		X					
BCGetAttributes		X	X	X	X	X	X
BCPostOnReceipt			X				
BCPrepareToReceive		X					
BCReceiveAndWait		X	X				
BCReceiveImmediate			X				
BCRequestToSend			X	X			
BCSendData		X					
BCSendError		X	X	X	X	X	
BCTest†							
posted			X				
request to send		X	X				

\* The state from which the routine can be executed varies depending on the type of deallocation; see "BCDeallocate," later in this chapter.

† The state from which the routine can be executed varies depending on the type of testing; see "BCTest," later in this chapter.



---

## Data mapping

Over an APPC mapped conversation, data sent between two transaction programs is a stream of data bytes that is divided into logical data records. Each time data is sent using the `MCSendData` routine, it is packaged into a single data record. The `MCReceiveAndWait` routine can read part or all of this data record. The routine interface imposes the structure of individual data records on the data byte stream.

At the data record level, however, the data may still need further transformation before it can be used by the transaction program. For example, the data may need to be formatted into a data structure of a high-level language. To insulate the transaction program from the process of transforming data between the form used by the transaction program and the transmitted byte stream, the LU 6.2 mapped conversation protocol defines **data mapping** in the LU to perform this function.

When data is sent, a map name is specified that identifies a mapping function, which transforms the data from the form used by the transaction program into the data stream sent to the remote LU. The current map name is sent to the remote LU, which uses it to execute a mapping function in the remote LU to transform the received data stream into the form used by the remote transaction program.

The map name specified by the local program may be different from the map name received by the remote program. The map name may be translated by the sending LU from a local map name known to the local transaction program into a global map name known to the remote LU. The remote LU may in turn translate the received map name into a map name known locally to the remote transaction program on the remote system.

Data mapping is not required on a mapped conversation. A null map name specifies no data mapping. A null map name is never translated into a nonnull map name (but a nonnull map name may be translated to a null map name, thus disabling mapping).

Under the LU 6.2 architecture, map name transmission, as well as data transformation, is handled by a user-supplied mapping utility, or **mapper**. At the time of publication, the MacAPPC mapper is limited to data transformation alone. The transmission of map names between the local and remote LUs is handled by the MacAPPC Conversation Driver.

---

## Writing a mapping procedure

To write a mapping procedure, you will need to use the mapping parameter block.

### Mapping parameter block

```
APPCMCPB =          RECORD
    mcpbMapCmd       : SignedByte;    { MC request }
    mcpbResult       : INTEGER;        { mapper return code }
    mcpbMapName      : StringPtr;      { map name pointer }
    mcpbDataPtr      : Ptr;            { data pointer }
    mcpbDataSize     : INTEGER;        { data length }
    mcpbBuffPtr      : Ptr;            { buffer pointer }
    mcpbBuffSize     : INTEGER;        { buffer length }
    mcpbTransMapName : Boolean;         { map name translation required }
    mcpbFMHdrr       : Boolean;        { FMH data contains FM headers }
    mcpbRcvMode      : SignedByte;    { receive mode }
END;

APPCMCPBPtr = ^APPCMCPB;
```



The following values are constants for `mcpbMapCmd`:

```
kSendMapping      = 0;
kRcvMapping       = 1;
```

The following values are constants for `mcpbResult`:

```
mcNoErr           = 0;
mcErr             = 1;
mcMapNameErr      = 2;
mcDupMapNameErr   = 3;
```

The following values are constants for `mcpbRcvMode`:

```
kTruncMode        = 0;
kIncomplMode      = 1;
```

The mapper is executed by the MacAPPC Conversation Driver when send or receive data mapping is required. The driver passes the address of the `APPCMCPB` parameter block as its single argument. The mapper performs the operation as specified by the `APPCMCPB` record, updates that record, and returns to the driver.

The mapping function is not required to use the space provided by the `mcpbBuffPtr` and `mcpbBuffSize` parameters to map the data, but the driver reserves 4 bytes immediately ahead of `mcpbBuffPtr` and uses them when formatting the data record to be sent on the conversation. At this time, mappers that pass mapped data back in a location other than in the space provided by `mcpbBuffPtr` and `mcpbBuffSize` do so at their own risk.

When the driver receives a new map name from the partner LU, it is passed to the mapper on the next receive mapping request, and the `mcpbTransMapName` field is set to TRUE. The mapper is responsible for translating, if necessary, the global map name to a local map name known to the transaction program. When the current map name has not changed, it is repeatedly passed to the mapper with `mcpbTransMapName` set to FALSE. The mapper has to translate the original global name only once, as the translated map name becomes the current map name and is passed back to the mapper on subsequent receive mapping requests.

When mapping receive data, the mapper selects the receive mode for the data. When the receive mode is incomplete mode, the transaction program may receive the data record in more than one receive routine. Data that is not read by the transaction program on the first receive routine is held by the LU until the record is completely read. When the receive mode is truncate mode, any data that is not read by the transaction program on the first receive routine is discarded by the LU. The manner in which the mapper selects the mode that it is to use is left to the implementor of the mapper (except when no mapping is performed, in which case incomplete mode is the default). The LU 6.2 protocol allows the mapping function to select either mode based on the map name or the individual implementation.

The LU 6.2 protocol allows the optional check by the mapper for duplicate map names sent by the remote LU. When the mapper receives a request that sets the `mcpbTransMapName` parameter to TRUE for the same map name specified in a previous request, the mapper may optionally return a mapper error indicating a duplicate map name.



## Default mapping procedure

The following Pascal procedure—the default mapping procedure—illustrates a sample mapping procedure.

```
PROCEDURE DefaultMapper(VAR mcParam : APPCMCPB);
{ default mapper }
{ This is the Pascal equivalent of the default mapping procedure that }
{ is used when cvMapProc is set to NIL on an MCAAllocate }
{ or when tpMapProc is set to NIL on a TPAttach(kWaitAttach)}
{ Remember, mapping procedures must run at interrupt time. }
{ This means you can't call routines that may move memory, }
{ or reference global data (A5 may not be valid). }

VAR
    i:    INTEGER;
    s,t:  ^SignedByte;

BEGIN

    { Since this mapper does not look at the map name, it will      }
    { never return mcMapNameErr or mcDupMapNameErr.  It also does  }
    { not look at mcpbTransMapName, since it never performs any   }
    { translation on the map name, nor checks for duplicate map names. }
    mcParam.mcpbResult := mcNoErr; { no error }

    CASE mcParam.mcpbMapCmd OF
        kSendMapping:
            BEGIN
                { send mapping - copy data into buffer }
                s := POINTER(mcParam.mcpbDataPtr);
                t := POINTER(mcParam.mcpbBuffPtr);
                FOR i := 1 TO mcParam.mcpbDataSize DO
                    BEGIN
                        t^ := s^;
                        s := POINTER(ORD4(s) + 1);
                        t := POINTER(ORD4(t) + 1);
                    END;
                mcParam.mcpbDataPtr := mcParam.mcpbBuffPtr;
            END;

        kRcvMapping:
            { receive mapping - since no mapping is done, }
            { set truncate data receive mode }
            mcParam.mcpbRcvMode := kIncomplMode;

        OTHERWISE
            { map execution failure }
            mcParam.mcpbResult := mcErr;
    END; { CASE }

END; { DefaultMapper }
```



---

---

## MacAPPC conversation routines

The next sections describe the MacAPPC conversation routines, which you use to communicate between a pair of transaction programs over a conversation. A conversation can be either mapped or basic.

The routines available for communicating over a conversation are divided into the following categories:

- ☐ mapped conversation routines, which are used to communicate over mapped conversations
- ☐ type-independent conversation routines, which can be used over either mapped or basic conversations
- ☐ basic conversation routines, which are normally used only over basic conversations, but can be used over mapped conversations

---

---

## Mapped conversation routines

This section describes the MacAPPC mapped conversation routines, which are used by transaction programs to communicate over mapped conversations. When a transaction program sends data by way of a mapped routine, the transaction program does not have to provide the General Data Stream (GDS) length field in front of the data. Instead, the data placed into the send buffer is dealt with as a complete data record, and the mapped conversation routines perform the necessary translation of data formatted as logical records into GDS variables.

In addition, the transaction program has the option of performing additional mapping functions on the data.



---

## MCAAllocate

### Summary

The `MCAAllocate` routine allocates a session between the local LU and a remote LU, and, within that session, a mapped conversation between the local transaction program and a remote transaction program. The routine also returns a **conversation ID**, which is used to identify the conversation.

---

### Important

If the local transaction program is starting a mapped conversation, it must execute a `MCAAllocate` routine before it executes any other mapped conversation routine.

If the local transaction program is waiting for a remote transaction program to start the mapped conversation, the local transaction program must execute a `TPAttach` routine. See `TPAttach` in Chapter 7 of this manual.

---

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0034	long	→	cvMapBuffPtr
0038	word	→	cvMapBuffSize
002E	long	→	cvPIPBuffPtr
0032	word	→	cvPIPBuffSize
0042	long	→	cvRmtLUName
0052	long	→	cvRmtProgName
004E	long	→	cvModeName
0056	long	→	cvUserName
005A	long	→	cvUserPswd
005E	long	→	cvUserProf
007A	long	→	cvMapProc
008A	byte	→	cvReturnCtl
0089	byte	→	cvSyncType
008F	byte	→	cvPIPUsed
0492	----	→	cvPIPPtr[]
0892	----	→	cvPIPSize[]
008B	byte	↔	cvSecType
003A	long	←	cvConvID

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to a Conversation Control Block (CVCB) whose length is determined by the value of the `kCVCBSize` constant. You must supply a new CVCB each time your application executes an `MCAAllocate` routine.



**cvMapBuffPtr** (supplied) specifies a pointer to a mapped conversation buffer. The length of the buffer is specified by the value of the **cvMapBuffSize** parameter.

**cvMapBuffSize** (supplied) specifies the size of the mapped conversation buffer pointed to by the **cvMapBuffPtr** parameter. This buffer must be large enough to hold the largest complete data record expected plus a 4-byte logical length ID (LLID).

**cvPIPBuffPtr** (supplied) specifies a pointer to a buffer that holds the program initialization parameters. The length of the buffer is specified by the value of the **cvPIPBuffSize** parameter.

**cvPIPBuffSize** (supplied) specifies the size of the buffer pointed to by **cvPIPBuffPtr**. This buffer must be large enough to hold the largest amount of PIP data expected plus a 4-byte LLID per parameter plus one 4-byte LLID for the entire PIP data.

**cvRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU. The string length must not be greater than the value of the **kMaxName** constant. The name is any name by which the local LU knows the remote LU for the purpose of allocating a mapped conversation. This locally known LU name becomes the LU name that is used by the network if the two names are different.

**cvRmtProgName** (supplied) specifies a pointer to a string that contains the name of the remote transaction program at the remote LU specified by the **cvRmtLUName** parameter. The string length must not be greater than the value of the **kMaxTPName** constant. For mapped conversations, the string cannot specify an SNA service transaction program.

**cvModeName** (supplied) specifies a pointer to a string that contains the name of the mode defining certain properties for the session allocated to the conversation. The string length must not be greater than the value of the **kMaxName** constant. The properties that are defined include, for example, class of service to be used, and whether data is to be enciphered or translated into ASCII before it is sent. The SNA-defined mode name **SNASVCMG** must not be specified for the **MCAAllocate** routine (whereas the **BCAllocate** routine can use that mode name; see the description of that routine in this chapter).

**cvUserName** (supplied) specifies a pointer to a string that contains the user ID when the **cvSecType** parameter has the value of the **kProgSec** constant (otherwise, the parameter is ignored). The string length must not be greater than the value of the **kMaxSecName** constant. The remote LU uses this value and the password to verify the identity of the transaction program making the allocation request. In addition, the remote LU can use the **cvUserName** parameter for auditing or accounting purposes, or it can use **cvUserName**, together with the profile (see **cvUserProf**), to determine which remote transaction programs the local transaction program can access and which resources the remote transaction program can access.

**cvUserPswd** (supplied) specifies a pointer to a string that contains the password when the **cvSecType** parameter has the value of the **kProgSec** constant (otherwise, the parameter is ignored). The string length must not be greater than the value of the **kMaxSecName** constant. The remote LU uses this value and the value specified in the **cvUserName** parameter to verify the identity of the transaction program making the allocation request.



**cvUserProf** (supplied) specifies a pointer to a string that can contain a profile to be used in place of or in addition to the user ID specified in the `cvUserName` parameter. The string length must not be greater than the value of the `kMaxSecName` constant. The remote LU can use this value, in addition to or in place of the value specified in the `cvUserName` parameter, to determine which remote transaction programs the local transaction program can access, and which resources the remote transaction program can access.

**cvMapProc** (supplied) specifies a pointer to a user-supplied mapping function (see the section "Writing a Mapping Procedure," earlier in this chapter, for a description of the mapping function). Set the pointer to NIL to use the default mapping procedure.

**cvReturnCtl** (supplied) specifies when the local LU is to return control to the transaction program and what type of session allocation is to be used. If the local LU fails to obtain a session for the mapped conversation, an allocation error is reported either on this routine or on a subsequent routine. If the remote LU rejects the allocation request, an allocation error is reported on a subsequent routine. The following values are defined:

`kWhenAllocReturn` allocates a session before returning control to the local transaction program. A session-allocation error is reported upon return from the `MCAAllocate` routine.

`kDelayAllocReturn` allocates a session after returning control to the local transaction program. A session-allocation error is reported upon return from a subsequent `MacAPPC` routine.

`kImmedAllocReturn` allocates a session only if a session is immediately available and returns control to the local transaction program. A session is immediately available when it is a free first-speaker session. A session-allocation error is reported upon return from the `MCAAllocate` routine if a session is not immediately available.

**cvSyncType** (supplied) specifies the synchronization level that the local and remote transaction programs can use for the conversation. The values are defined as follows:

`kNoSync` specifies that the transaction programs do not perform confirmation processing nor sync-point processing on this mapped conversation. The transaction programs do not execute any routines and do not recognize any returned parameters relating to confirmation or synchronization functions.

`kConfirmSync` specifies that transaction programs can perform confirmation processing but not sync-point processing on this mapped conversation. The transaction programs do not execute any routines and do not recognize any returned parameters relating to the synchronization functions.

`kSyncPtSync` specifies that transaction programs can perform both confirmation processing and sync-point processing on this mapped conversation.

❖ *Note:* At the time of publication, sync-point services were not supported.

**cvPIPUsed** (supplied) specifies whether or not program initialization parameters (PIPs) are to be sent to the remote transaction program. A value of TRUE specifies that PIP data is present; FALSE specifies that PIP data is not present.



**cvPIPPtr** (supplied) specifies an array of pointers to program initialization parameters. The last pointer must be followed by one that is NIL. The maximum number of parameters is defined by the value of the **kMaxPIP** constant, with a total space limitation specified by the **cvPIPBuffSize** parameter (see **cvPIPBuffSize** for more information about space limitations). This array is ignored if the **cvPIPUsed** parameter is set to FALSE.

**cvPIPSize** (supplied) specifies an array of sizes that specifies the size for each PIP in the **cvPIPPtr** array. The last size must be followed by a size of 0.

**cvSecType** (supplied/returned) specifies the type of access-security information that is to be used by the remote LU to validate access to the remote transaction program and its resources. The following values are defined:

**kNoSec** specifies that access-security information is not to be used.

**kSameSec** specifies that the security information to be used is from the local transaction program executing the **MCAAllocate** routine, so that the security level remains the same as set by the previous allocation request. The allocation request carries the user name of the local transaction program and is indicated as already verified (that is, no password is sent). If the local transaction program was not previously allocated, the **cvSecType** parameter is downgraded to the value of the **kNoSec** constant.

**kProgSec** specifies that the security information to be used is contained in the **cvUserName**, **cvUserPswd**, and optionally the **cvUserProf** parameters.

**cvConvID** (returned) indicates the conversation ID of the allocated conversation.

## Notes

Successful completion of the **MCAAllocate** routine does not indicate that the session was successfully allocated. This routine can only return session-allocation errors (with the **cvReturnCtl** parameter equal to the value of the **kWhenAllocReturn** or **kImmedAllocReturn** constant). All other allocation errors are reported on subsequent routines.

When control returns and no error was encountered, the conversation is in send state.

For IBM equipment, make sure that the PIP data is in the format that the receiving transaction program expects. For example, you may need to execute the **TPAsciiToEbcdic** routine.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **BCAllocate**, **MCDeallocate**



---

## MCConfirm

<b>Summary</b>	The <code>MCConfirm</code> routine flushes the send buffer, transmits a request for confirmation to the remote transaction program, and waits for a reply. The remote transaction program replies with either a confirmation or an error. This routine allows the local and remote transaction programs to synchronize their processing. This routine is not available on conversations allocated with a synchronization level of none.			
<b>Parameters</b>	000C	long	→	<code>ioCompletion</code>
	0018	word	→	<code>appcRefNum</code>
	001A	word	→	<code>appcOpCode</code>
	0022	long	→	<code>appcUserRef</code>
	0010	word	←	<code>ioResult</code>
	001C	word	←	<code>appcHiResult</code>
	001E	word	←	<code>appcLoResult</code>
	0020	byte	←	<code>appcConvState</code>
	0026	long	→	<code>cvTPCBPtr</code>
	002A	long	→	<code>cvCVCBPtr</code>
	0082	byte	←	<code>cvReqToSendRcvd</code>
<b>Description</b>	<p><b><code>cvTPCBPtr</code></b> (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).</p> <p><b><code>cvCVCBPtr</code></b> (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).</p> <p><b><code>cvReqToSendRcvd</code></b> (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.</p>			
<b>Result code</b>	<code>appcNoErr</code>	Routine succeeded		
	<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>		
	<code>appcExec</code>	Routine executing; asynchronous request not complete		
<b>See also</b>	<code>BCConfirm</code> , <code>MCConfirmed</code>			



---

## MCConfirmed

<b>Summary</b>	The MCConfirmed routine sends a confirmation reply to the remote transaction program when a confirmation request is received.			
<b>Parameters</b>	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0020	byte	←	appcConvState
	0026	long	→	cvTPCBPtr
	002A	long	→	cvCVCBPtr
<b>Description</b>	<b>cvTPCBPtr</b> (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).			
	<b>cvCVCBPtr</b> (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).			
<b>Notes</b>	When control returns and no error is encountered, the conversation state changes as follows: if the conversation was in confirm state, it goes to receive state; if the conversation was in confirm/send state, it goes to send state; if the conversation was in confirm/deallocate state, it goes to deallocate state.			
<b>Result code</b>	appcNoErr	Routine succeeded		
	appcFail	Routine failed; look in appcHiResult and appcLoResult		
	appcExec	Routine executing; asynchronous request not complete		
<b>See also</b>	BCConfirmed, MCConfirm			



---

## MCDeallocate

### Summary

The `MCDeallocate` routine flushes the send buffer and deallocates the mapped conversation from the transaction program. It can also include the function of the `MCConfirm` routine.

---

### Important

Your transaction program must execute a `MCDeallocate` routine to end a mapped conversation. After the `MCDeallocate` routine has been executed, no more mapped conversation routines can be executed for that deallocated mapped conversation.

---

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0085	byte	→	cvDeallocType

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB) to be deallocated.

**cvDeallocType** (supplied) specifies the type of deallocation:

**kSyncDealloc** specifies that MacAPPC should perform deallocation based on the synchronization level allocated to this mapped conversation. A synchronization level of none performs a deallocation as if the **kFlushDealloc** constant had been specified. A synchronization level of confirm performs a deallocation as if the **kConfirmDealloc** constant had been specified.

**kFlushDealloc** specifies that MacAPPC should execute the function of the `MCFlush` routine and deallocate the conversation normally.

**kConfirmDealloc** specifies that MacAPPC should execute the function of the `MCConfirm` routine, and if it is successful, deallocate the conversation normally; if it is not successful, the state of the conversation is determined by the result code.

**kAbendDealloc** specifies that MacAPPC should execute the function of the `MCFlush` routine when the conversation is in send or defer state, and deallocate the conversation normally. If the conversation is in receive state, data can be lost. **kAbendDealloc** is intended to be used by a transaction program when it detects an error condition that prevents completion of the transaction.



`kLocalDealloc` specifies that MacAPPC should deallocate the mapped conversation locally. The transaction program should specify this type of deallocation if, and only if, the conversation is in deallocate state.

## Notes

When control returns, if no errors were encountered, the conversation enters reset state. When the `cvDeallocType` parameter is set to the `kSyncDealloc` constant, and the partner sends an error response, the conversation enters receive state.

The execution of the `MCFlush` or `MCConfirm` routine as part of the `MCDeallocate` routine includes the flushing of the LU's send buffer. When, instead, the deallocation is deferred, the LU also defers flushing its send buffer until the program executes a subsequent routine for this conversation.

## Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

## See also

`BCDeallocate`, `MCAAllocate`



---

## MCFlush

### Summary

The MCFlush routine sends the information that is in the LU's send buffer to the remote transaction program. Information is buffered in the send buffer by the MCAAllocate, MCDeallocate, MCSendData, and MCSendError routines.

The MCFlush routine is useful for optimization of processing between the local and remote transaction programs. The LU normally buffers the data records from consecutive MCSendData routines until it has a sufficient amount for transmission. At that time it transmits the buffered data records. However, the local transaction program can execute an MCFlush routine in order to cause the LU to transmit the buffered data records. In this way, the local transaction program can minimize the delay in the remote transaction program's processing of the data records.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

### Notes

When the routine completes without error, no state change occurs. If an error is detected, the conversation enters either receive state or deallocate state, depending on the error.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

BCFlush



---

## MCGetAttributes

**Summary** The MCGetAttributes routine returns information about the specified mapped conversation.

**Parameters**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
004A	long	⇒	cvFullLclLUName
0042	long	⇒	cvRmtLUName
0046	long	⇒	cvFullRmtLUName
004E	long	⇒	cvModeName
0056	long	⇒	cvUserName
005E	long	⇒	cvUserProf
0062	long	⇒	cvLUWName
0066	long	⇒	cvLUWID
006A	long	⇒	cvLUWCorr
0089	byte	←	cvSyncType
006E	word	←	cvLUWSeq
003A	long	←	cvConvID
003E	long	←	cvProgID

**Description** **cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvFullLclLUName** (supplied/modified) specifies a pointer to space where the fully qualified network name of the local LU can be returned. The space must be at least two times the value of the `kMaxName` constant plus 1 byte. The name is returned as two strings concatenated by a period: NETID.NETNAME, where NETID is the network ID and NETNAME is the network LU name. When there is no network ID, only NETNAME is returned. If the pointer is NIL, the name is not returned.

**cvRmtLUName** (supplied/modified) specifies a pointer to space where the local name of the remote LU—that is, where the remote transaction program is located—can be returned. The space must be at least as large as the value of the `kMaxName` constant plus 1 byte. This is the name by which the local LU knows the remote LU for the purpose of allocating a conversation. If the pointer is NIL, the name is not returned.

**cvFullRmtLUName** (supplied/modified) specifies a pointer to space where the fully qualified network name for the remote LU can be returned. The space must be at least two times the value of the `kMaxName` constant plus 1 byte. The name has the same form as that specified in the `cvFullLclLUName` parameter. If the remote LU's fully qualified name is not known, a NULL string is returned. If the pointer is NIL, the name is not returned.



**cvModeName** (supplied/modified) specifies a pointer to space where the mode name for the session allocated to the conversation can be returned. The space must be at least as large as the value of the `kMaxName` constant plus 1 byte. If the pointer is NIL, the name is not returned.

**cvUserName** (supplied/modified) specifies a pointer to space where the user ID can be returned. The space must be at least as large as the value of the `kMaxSecName` constant plus 1 byte. The remote LU uses this value and the password to verify the identity of the transaction program making the allocation request. In addition, the remote LU can use the value of the `cvUserName` parameter for auditing or accounting purposes, or it can use the value of `cvUserName`, together with the profile (see `cvUserProf`), to determine which remote transaction programs the local transaction program can access and which resources the remote transaction program can access. If the pointer is NIL, the user ID is not returned.

**cvUserProf** (supplied/modified) specifies a pointer to space where a profile to be used in place of or in addition to the user ID specified in the `cvUserName` parameter can be returned. The space must be at least as large as the value of the `kMaxSecName` constant plus 1 byte. The remote LU can use this value, in addition to or in place of `cvUserName`, to determine which remote transaction programs the local transaction program can access and which resources the remote transaction program can access. If the pointer is NIL, the profile is not returned.

**cvLUWName** (supplied/modified) specifies a pointer to space where the LU name portion of the logical-unit-of-work (LUW) can be returned. The space must be at least as large as the value of the `kMaxLUWName` constant plus 1 byte. The LUW identifier is created and maintained by the LU, which uses it for accounting purposes. If the pointer is NIL, the name is not returned.

**cvLUWID** (supplied/modified) specifies a pointer to space where the unique ID portion of the logical-unit-of-work can be returned. The space must be at least as long as the value of the `kMaxLUWID` constant plus 1 byte. This is not a printable string. If the pointer is NIL, the LUW ID is not returned.

**cvLUWCorr** (supplied/modified) specifies a pointer to space where the conversation correlator can be returned. The space must be at least as long as the value of the `kMaxLUWCorr` constant plus 1 byte. The conversation correlator is created and maintained by the LU. This is not a printable string. If the pointer is NIL, the name is not returned.

**cvSyncType** (returned) indicates the synchronization level that the local and remote transaction programs can use for the conversation. The following values are defined:

`kNoSync` indicates that the transaction programs do not perform confirmation processing on this mapped conversation. The transaction programs do not execute any routines and do not recognize any returned parameters relating to these synchronization functions.

`kConfirmSync` indicates that transaction programs can perform confirmation processing but not sync-point processing on this mapped conversation. The transaction programs do not execute any routines and do not recognize any returned parameters relating to the synchronization functions.

`kSyncPtSync` indicates that transaction programs can perform both confirmation processing and sync-point processing on this mapped conversation.

❖ *Note:* At the time of publication, sync-point services were not supported.



**cvLUWSeq** (returned) is the logical-unit-of-work sequence number assigned to this conversation.

**cvConvID** (returned) returns the conversation ID.

**cvProgID** (returned) returns the program ID.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **BCGetAttributes**



---

## MCPPostOnReceipt

### Summary

The `MCPPostOnReceipt` routine requests a mapped conversation to be posted when information is available for the transaction program to receive. Execute a `MCTest` routine after `MCPPostOnReceipt` to determine if posting has occurred. Execute a `CVWait` routine after `MCPPostOnReceipt` to wait for posting to occur.

### Parameters

000C	long	→	<code>ioCompletion</code>
0018	word	→	<code>appcRefNum</code>
001A	word	→	<code>appcOpCode</code>
0022	long	→	<code>appcUserRef</code>
0010	word	←	<code>ioResult</code>
001C	word	←	<code>appcHiResult</code>
001E	word	←	<code>appcLoResult</code>
0020	byte	←	<code>appcConvState</code>
0026	long	→	<code>cvTPCBPtr</code>
002A	long	→	<code>cvCVCBPtr</code>
0074	word	→	<code>cvDataSize</code>

### Description

**`cvTPCBPtr`** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**`cvCVCBPtr`** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**`cvDataSize`** (supplied) specifies the minimum amount of data that will cause posting to occur. The `cvDataSize` parameter must be small enough so that the amount of data that will cause posting does not exceed the size of the LU's receive buffer. If that buffer is exceeded, posting will never occur. The proper size for the LU's receive buffer depends upon several factors; you will have to empirically test different values for `cvDataSize` to find the correct value.

❖ *Note:* If a small value has been specified for `cvDataSize` and a large data block has begun to arrive, you can subsequently use either an `MCTReceiveAndWait` routine to receive the entire data block or an `MCTReceiveImmediate` routine to receive only the data that has been received by the LU.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`BCPostOnReceipt`, `MCTest`, `CVWait`



---

## MCPPrepareToReceive

### Summary

The `MCPPrepareToReceive` routine flushes the send buffer and puts the mapped conversation into receive state. This routine can also include the function of the `MCCConfirm` routine, requesting confirmation before entering receive state.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0086	byte	→	cvPrepToRcvType
0087	byte	→	cvLockType

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvPrepToRcvType** (supplied) specifies the type of request:

`kFlushRcv` indicates that the send buffer should be flushed (as in the `MCFlush` routine) and the conversation should enter receive state.

`kConfirmRcv` indicates that the data in the buffer should be sent and that confirmation will be requested before entering receive state.

**cvLockType** (supplied) specifies when control is to be returned to the local transaction program if the function of the `MCCConfirm` routine is performed. This parameter is ignored if the mapped conversation does not support confirmation processing.

`kShortLock` specifies that control should be returned to the local transaction program when an affirmative reply is received.

`kLongLock` specifies that control should be returned to the local transaction program when data following confirmation is received.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`BCPrepareToReceive`



---

## MCRceiveAndWait

### Summary

The `MCRceiveAndWait` routine receives information that has arrived for the mapped conversation. The routine can wait for more information to arrive if the request is not yet satisfied. The information can be data, conversation status, or a request for confirmation. The program can execute this routine when the conversation is in send state, which flushes the buffer and places the conversation in receive state.

The program receives only one kind of information at a time; that is, it can receive data or control information, but it cannot receive both simultaneously.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0070	long	⇒	cvDataPtr
0074	word	↔	cvDataSize
0076	long	⇒	cvMapName
0084	byte	←	cvWhatRcvd
0082	byte	←	cvReqToSendRcvd

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvDataPtr** (supplied/modified) specifies a pointer to space into which received data is copied. The size of the space is specified in the `cvDataSize` parameter.

**cvDataSize** (supplied/returned) specifies the maximum amount of data the program is to receive. When control returns to the program, this field contains the length of data received.

**cvMapName** (supplied/modified) specifies a pointer to space where the local map name can be returned. The space must be at least as large the value of the `kMaxMapName` constant plus 1 byte. A NULL string indicates that no data mapping occurred. When the program receives information other than data, nothing is placed in this variable.

❖ *Note:* The space pointed to by this parameter is overwritten each time your application executes an `MCRceiveAndWait` routine. Your application must save the name if necessary.

**cvWhatRcvd** (returned) indicates what was received.

`kDataComplRcvd` indicates that a complete logical length record (LL) or the end of a previously incomplete LL was received.



kDataIncomplRcvd indicates that an incomplete logical length record was received, and the LU retained the remainder of the data record. The program can receive the remainder of the data record by executing one or more additional MCRceiveAndWait routines.

kLLTruncRcvd indicates that a complete logical length record was not received, and the rest of the data was discarded. No data is returned.

kFMHDataComplRcvd indicates that the complete or last portion of the data record containing FMH data was received.

kFMHDataIncomplRcvd indicates that an incomplete data record containing FMH data was received, and the remainder was retained by the LU. The program can receive the remainder of the data record by executing one or more additional MCRceiveAndWait routines.

kFMHDataTruncRcvd indicates an incomplete data record containing FMH data was received. The remainder is discarded by the LU.

kSendRcvd indicates that the remote transaction program has entered receive state and sent a send indication. No data is returned. The conversation enters send state. The local transaction program can now execute an MCSendData routine.

kConfirmRcvd indicates that the remote transaction program has requested confirmation. No data is returned. The conversation enters confirm state.

kConfirmSendRcvd indicates that the remote transaction program has executed a prepare-to-receive confirm. The conversation enters confirm/send state.

kConfirmDeallocRcvd indicates that the remote transaction program has executed a deallocate confirm. The conversation enters confirm/deallocate state.

**cvReqToSendRcvd** (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

## Notes

When the amount of data requested is less than the length of the data record, whether or not the remainder of the record is discarded or retained is determined by the mapping procedure.

When control returns and no errors are encountered, the conversation is in receive state unless otherwise indicated by the cvWhatRcvd parameter.

For IBM equipment, be aware of the format that the sending TP is using. For example, you may need to execute the TPEbcdicToAscii routine.

## Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

## See also

BReceiveAndWait, MCRceiveImmediate



---

## MCRceiveImmediate

### Summary

The `MCRceiveImmediate` routine receives information that is available for the mapped conversation, but does not wait for information to arrive. The information can be application data, FMH data, or conversation control information. If an error is detected, the appropriate result code is returned. If there is nothing to receive, the routine returns a result code indicating an unsuccessful completion (`badComplErr`). Control is returned to the program with an indication of whether any information was received and, if so, the type of information.

Only data that has been processed by the MacAPPC drivers can be received; that is, the LU may have received data, but until the MacAPPC drivers process the data, it is not available to `MCRceiveImmediate` (this is not the case for the `MCRceiveAndWait` routine). The MacAPPC drivers process conversation data during the `MCRceiveAndWait`, `MCTest`, and `CVWait` routines.

When the `MCRceiveAndWait` routine (or a previous `MCRceiveImmediate` routine) returns the `kDataIncomplRcvd` or `kFMHDataIncomplRcvd` constant in the `cvWhatRcvd` parameter, `MCRceiveImmediate` can be used to receive the remaining data. Similarly, when the `MCTest` routine returns a result code of `dataAvail` or `ctlAvail`, or when the `CVWait` routine indicates posting for the conversation, `MCRceiveImmediate` can be used to receive data or control information (or possibly an error indication).

The program receives only one kind of information at a time; that is, it can receive data or control information, but it cannot receive both simultaneously.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0070	long	⇒	cvDataPtr
0074	word	↔	cvDataSize
0076	long	⇒	cvMapName
0084	byte	←	cvWhatRcvd
0082	byte	←	cvReqToSendRcvd

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvDataPtr** (supplied/modified) specifies a pointer to space into which received data is copied. The size of the space is specified in the `cvDataSize` parameter.

**cvDataSize** (supplied/returned) specifies the maximum amount of data the program is to receive. When control returns to the program, this field contains the length of data received.



**cvMapName** (supplied/modified) specifies a pointer to space where the local map name can be returned. The space must be at least as large the value of the **kMaxMapName** constant plus 1 byte. A NULL string indicates that no data mapping occurred. When the program receives information other than data, nothing is placed in this variable.

- ❖ *Note:* The space pointed to by this parameter is overwritten each time your application executes an **MCRReceiveImmediate** routine. Your application must save the name if necessary.

**cvWhatRcvd** (returned) indicates what was received, as follows:

**kDataComplRcvd** indicates that a complete logical length record (LL) or the end of a previously incomplete LL was received.

**kDataIncomplRcvd** indicates that an incomplete LL was received, and the LU retained the remainder of the data record. The program can receive the remainder of the data record by executing one or more additional **MCRReceiveImmediate** routines.

**kLLTruncRcvd** indicates that a complete logical length record was not received, and the rest of the data was discarded. No data is returned.

**kFMHDataComplRcvd** indicates that the complete or last portion of the data record containing FMH data was received.

**kFMHDataIncomplRcvd** indicates that an incomplete data record containing FMH data was received, and the remainder was retained by the LU. The program can receive the remainder of the data record by executing one or more additional **MCRReceiveAndWait** routines.

**kFMHDataTruncRcvd** indicates an incomplete data record containing FMH data was received. The remainder is discarded by the LU.

**kSendRcvd** indicates that the remote transaction program has entered receive state and sent a send indication. No data is returned. The conversation enters send state. The local transaction program can now execute an **MCSendData** routine.

**kConfirmRcvd** indicates that the remote transaction program has requested confirmation. No data is returned. The conversation enters confirm state.

**kConfirmSendRcvd** indicates that the remote transaction program has issued a prepare-to-receive confirm. The conversation enters confirm/send state.

**kConfirmDeallocRcvd** indicates that the remote transaction program has issued a deallocate confirm. The conversation enters confirm/deallocate state.

**cvReqToSendRcvd** (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

## Notes

When the amount of data requested is less than the length of the data record, whether or not the remainder of the record is discarded or retained is determined by the mapping procedure.



When control returns and no errors are encountered, the conversation is in receive state unless otherwise indicated by the `cvWhatRcvd` parameter.

For IBM equipment, be aware of the format that the sending transaction program is using. For example, you may need to execute the `TPEbcdicToAscii` routine.

<b>Result code</b>	<code>appcNoErr</code>	Routine succeeded
	<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
	<code>appcExec</code>	Routine executing; asynchronous request not complete

<b>See also</b>	<code>BCReceiveImmediate</code> , <code>MReceiveAndWait</code>
-----------------	--



---

## MCRequestToSend

### Summary

The `MCRequestToSend` routine is used by the local transaction program to request to enter send state. A request-to-send indicator is sent to the remote transaction program which can either honor or ignore the request. If the request is honored, the conversation enters send state for the local transaction program when a send indication is received from the remote transaction program.

### Parameters

000C	long	→	<code>ioCompletion</code>
0018	word	→	<code>appcRefNum</code>
001A	word	→	<code>appcOpCode</code>
0022	long	→	<code>appcUserRef</code>
0010	word	←	<code>ioResult</code>
001C	word	←	<code>appcHiResult</code>
001E	word	←	<code>appcLoResult</code>
0020	byte	←	<code>appcConvState</code>
0026	long	→	<code>cvTPCBPtr</code>
002A	long	→	<code>cvCVCBPtr</code>

### Description

**`cvTPCBPtr`** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**`cvCVCBPtr`** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

### Notes

No state change occurs.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`BCRequestToSend`



---

## MCSendData

### Summary

The `MCSendData` routine sends one data record to the remote transaction program.

The data is buffered in the local LU's send buffer and is not transmitted to the remote transaction program until the send capacity is exceeded. Transmission of the contents of the send buffer is forced by the `MCFlush` routine or by routines that include the flush function (for example, `MCPprepareToReceive`).

To support the capability of earlier LU 6 types, MacAPPC allows FM headers to be sent to the partner program. The FM headers are sent using `MCSendData` with the `cvFMHdrs` parameter set to `TRUE`. This indicates FM data rather than application data is being sent. This is significant only to the partner transaction program; the sending and receiving LUs perform no special FM header processing other than indicating that the data record contains FM headers (see `MCReceiveAndWait` or `MCReceiveImmediate`).

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0070	long	→	cvDataPtr
0074	word	→	cvDataSize
0076	long	→	cvMapName
0083	byte	→	cvFMHdrs
0082	byte	←	cvReqToSendRcvd

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvDataPtr** (supplied) points to the data to be sent. The size of the data is specified by the `cvDataSize` parameter. The data is sent as one complete data record, regardless of actual data format. Thus, no data formatting is required (as it is for data sent on a basic conversation).

**cvDataSize** (supplied) contains the length of the data to be sent. If this parameter is zero, a null data record is sent.

**cvMapName** (supplied) specifies a pointer to a string that contains the local map name. The string length must not be greater than the value of the `kMaxName` constant. If a name is specified, the name and the data are passed to the mapping procedure for mapping before transmission. For more information on mapping, see the section "Writing a Mapping Procedure," earlier in this chapter. A `NIL` pointer indicates that no data mapping should occur.



**cvFMHdrs** (supplied) specifies whether FM headers are being sent in the data record.

**cvReqToSendRcvd** (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

#### Notes

When no error is encountered, the conversation remains in send state.

For IBM equipment, be aware of the format that the remote transaction program is expecting. For example, you may need to execute the `TPAsciiToEbcdic` routine.

#### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

#### See also

`BCSendData`



---

## MCSendError

### Summary

The `MCSendError` routine sends an error notification to the remote transaction program, indicating that the local transaction program has detected an application error. If the conversation is in send state, the send buffer is flushed. When the routine is completed with no errors, the conversation is in send state and the remote transaction program is in receive state.

### Parameters

000C	long	→	<code>ioCompletion</code>
0018	word	→	<code>appcRefNum</code>
001A	word	→	<code>appcOpCode</code>
0022	long	→	<code>appcUserRef</code>
0010	word	←	<code>ioResult</code>
001C	word	←	<code>appcHiResult</code>
001E	word	←	<code>appcLoResult</code>
0020	byte	←	<code>appcConvState</code>
0026	long	→	<code>cvTPCBPtr</code>
002A	long	→	<code>cvCVCBPtr</code>
0082	byte	←	<code>cvReqToSendRcvd</code>

### Description

`cvTPCBPtr` (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

`cvCVCBPtr` (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

`cvReqToSendRcvd` (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

### Notes

`MCSendError` resets or cancels posting. If posting is active and the conversation has been posted, posting is reset. If the conversation has not been posted, posting is cancelled.

If a race condition arises where both the local and remote transaction programs issue error indications, the program that was in receive state wins the race and the program that was in send state receives a program-error indication.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`BCSendData`



---

## MCTest

### Summary

The `MCTest` routine tests the specified mapped conversation to see whether posting has occurred or whether a request-to-send notification has been received. If a request-to-send has been received, the routine returns an `appcNoErr` result code. If posting has occurred, the routine returns `appcNoErr` and the `appcLoResult` is set to one of the following constants:

`dataAvail` indicates that data has arrived.

`ctlAvail` indicates that control information has arrived.

When testing for posting, posting must be previously activated for the mapped conversation with the `MCPPostOnReceipt` routine.

### Parameters

000C	long	→	<code>ioCompletion</code>
0018	word	→	<code>appcRefNum</code>
001A	word	→	<code>appcOpCode</code>
0022	long	→	<code>appcUserRef</code>
0010	word	←	<code>ioResult</code>
001C	word	←	<code>appcHiResult</code>
001E	word	←	<code>appcLoResult</code>
0020	byte	←	<code>appcConvState</code>
0026	long	→	<code>cvTPCBPtr</code>
002A	long	→	<code>cvCVCBPtr</code>
008E	byte	→	<code>cvTestType</code>

### Description

**`cvTPCBPtr`** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**`cvCVCBPtr`** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**`cvTestType`** (supplied) specifies the condition to be tested, as follows:

`kPostTest` specifies to test whether the conversation has been posted.

`kReqToSendTest` specifies to test whether request-to-send notification has been received from the remote transaction program.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`BCTest`, `MCPPostOnReceipt`



---

---

## **Type-independent conversation routines**

This section describes the MacAPPC type-independent conversation routines. These routines can be used for either mapped or basic conversations. In addition, some of these routines can be executed for multiple conversations of different conversation types.



---

## CVBackout

### Summary

CVBackout is currently not supported.



---

## CVGetType

<b>Summary</b>	The CVGetType routine returns the conversation type (basic or mapped). The type is set when the conversation is allocated.			
<b>Parameters</b>	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0020	byte	←	appcConvState
	0026	long	→	cvTPCBPtr
	002A	long	→	cvCVCBPtr
	008C	byte	←	cvConvType
<b>Description</b>	<p><b>cvTPCBPtr</b> (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).</p> <p><b>cvCVCBPtr</b> (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).</p> <p><b>cvConvType</b> (returned) is set to the specified conversation's type. The following values can be returned:</p> <p>    kBasicConv specifies basic conversation type.</p> <p>    kMappedConv specifies mapped conversation type.</p>			
<b>Result code</b>	appcNoErr	Routine succeeded		
	appcFail	Routine failed; look in appcHiResult and appcLoResult		
	appcExec	Routine executing; asynchronous request not complete		
<b>See also</b>	MCAllocate, BCAllocate			



---

## CVSyncPoint

**Summary**      CVSyncPoint is currently not supported.



---

## CVWait

### Summary

The `CVWait` routine waits for posting to occur on any of the specified conversations. Posting for a conversation occurs when posting is active for the conversation that satisfies the posting request parameters (specified by the `BCPostOnReceipt` or `MCPPostOnReceipt` routine). When used with `MCPPostOnReceipt` or `BCPostOnReceipt`, `CVWait` permits the local transaction program to receive data in synchronous fashion from multiple conversations.

If posting has occurred, the routine returns `appcNoErr` and the `appcLoResult` parameter is set to one of the following constants:

`dataAvail` indicates that data has arrived.

`ctlAvail` indicates that control information has arrived.

When a specified conversation has already been posted, the `CVWait` routine returns immediately.

### Parameters

000C	long	→	<code>ioCompletion</code>
0018	word	→	<code>appcRefNum</code>
001A	word	→	<code>appcOpCode</code>
0022	long	→	<code>appcUserRef</code>
0010	word	←	<code>ioResult</code>
001C	word	←	<code>appcHiResult</code>
001E	word	←	<code>appcLoResult</code>
0020	byte	←	<code>appcConvState</code>
0026	long	→	<code>cvTPCBPtr</code>
002A	long	→	<code>cvCVCBPtr</code>
0092	----	→	<code>cvCVCBList[]</code>
0090	word	↔	<code>cvCVCBIndex</code>

### Description

**`cvTPCBPtr`** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**`cvCVCBPtr`** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**`cvCVCBList`** (supplied) specifies a list of CVCB pointers that identify the conversations that must wait for posting to occur.

**`cvCVCBIndex`** (supplied/returned) specifies the number of entries in the `cvCVCBList` parameter, and returns either the index into the `cvCVCBList` array of the first conversation that has been posted or 0 to indicate that an error occurred on the `CVWait` request.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`MCPPostOnReceipt`, `BCPostOnReceipt`



---

---

## Basic conversation routines

Basic conversation routines are intended for use with LU services programs. The LU services programs can provide services or protocol boundaries for transaction programs. For example, the MacAPPC drivers use basic conversation routines to process the mapped conversation routines. Other types of LU services programs include IBM-designed programs such as the Document Interchange Architecture (DIA) and the CNOS (change-number-of-sessions) transaction programs.

Mapped routines handle data records, while basic routines handle logical records. What this means to you as a programmer is that basic routines require a header to be sent with each block of data. The header field must contain a 2-byte logical length record (LL) portion, which specifies the length of the logical record. In addition, ID information can also be included in the header.

The use of the header permits basic conversation routines to send data in a more efficient manner, because each logical record does not need to be complete in order to be sent. In other words, the send buffer can be flushed, even when it contains only portions of logical records, or it can accumulate a series of small records that it can send with a single execution of the routine.

Basic routines are also able to transmit and receive data that is formatted in any data stream, not just General Data Stream (GDS) format. They can process IBM-specified data streams, such as 3270 and 5250, or SNA data streams, as well as user-defined data streams.



---

## BCAllocate

### Summary

The `BCAllocate` routine allocates a session between the local LU and a remote LU, and, within the same session, a conversation between the local transaction program and a remote transaction program. A conversation ID is returned which is used to identify the conversation.

---

### Important

If the local transaction program is starting a basic conversation, it must execute a `BCAllocate` routine before it executes any other basic conversation routine.

If the local transaction program is waiting for a remote transaction program to start the basic conversation, the local transaction program must execute a `TPAttach` routine. See `TPAttach` in Chapter 7 of this manual.

---

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
002E	long	→	cvPIPBuffPtr
0032	word	→	cvPIPBuffSize
0042	long	→	cvRmtLUName
0052	long	→	cvRmtProgName
004E	long	→	cvModeName
0056	long	→	cvUserName
005A	long	→	cvUserPswd
005E	long	→	cvUserProf
008C	byte	→	cvConvType
008A	byte	→	cvReturnCtl
0089	byte	→	cvSyncType
008F	byte	→	cvPIPUsed
0492	----	→	cvPIPPtr[]
0892	----	→	cvPIPSize[]
008B	byte	↔	cvSecType
003A	long	←	cvConvID

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to a Conversation Control Block (CVCB) whose length is determined by the `kCVCBSize` constant. You must supply a new CVCB each time your application executes a `BCAllocate` routine.

**cvPIPBuffPtr** (supplied) specifies a pointer to a buffer that holds the program initialization parameters. The length of the buffer is specified by the value of the `cvPIPBuffSize` parameter.



**cvPIPBuffSize** (supplied) specifies the size of the buffer pointed to by **cvPIPBuffPtr**. This buffer must be large enough to hold the largest amount of PIP data expected plus a 4-byte LLID per parameter plus one 4-byte LLID for the entire PIP data.

**cvRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU. The string length must not be greater than the value of the **kMaxName** constant. The name is any name by which the local LU knows the remote LU for the purpose of allocating a basic conversation. This locally known LU name becomes the LU name that is used by the network if the two names are different.

**cvRmtProgName** (supplied) specifies a pointer to a string that contains the name of the remote transaction program at the remote LU specified by the **cvRmtLUName** parameter. The string length must not be greater than the value of the **kMaxTPName** constant. A transaction program that has the appropriate privilege can specify the name of an SNA service transaction program.

**cvModeName** (supplied) specifies a pointer to a string that contains the name of the mode defining certain properties for the session allocated to the conversation. The string length must not be greater than the value of the **kMaxName** constant. The properties that are defined include, for example, class of service to be used and whether data is to be enciphered or translated into ASCII before it is sent. The SNA-defined mode name **SNASVCMG** is reserved for LU service programs.

**cvUserName** (supplied) specifies a pointer to a string that contains the user ID when the **cvSecType** parameter is set to the **kProgSec** constant (otherwise, the parameter is ignored). The string length must not be greater than the value of the **kMaxSecName** constant. The remote LU uses this value and the password to verify the identity of the transaction program making the allocation request. In addition, the remote LU can use **cvUserName** for auditing or accounting purposes, or it can use **cvUserName**, together with the profile (see **cvUserProf**), to determine which remote transaction programs the local transaction program can access and which resources the remote transaction program can access.

**cvUserPswd** (supplied) specifies a pointer to a string that contains the password when the **cvSecType** parameter is set to the **kProgSec** constant (otherwise, the parameter is ignored). The string length must not be greater than the value of the **kMaxSecName** constant. The remote LU uses this value and the value specified in the **cvUserName** parameter to verify the identity of the transaction program making the allocation request.

**cvUserProf** (supplied) specifies a pointer to a string that contains a profile to be used in place of or in addition to the user ID specified in the **cvUserName** parameter. The string length must not be greater than the value of the **kMaxSecName** constant. The remote LU can use this value, in addition to or in place of the value specified in the **cvUserName** parameter, to determine which remote transaction programs the local transaction program can access, and which resources the remote transaction program can access.

**cvConvType** (supplied) specifies the conversation type, as follows:

**kBasicConv** for basic conversation.

**kMappedConv** for mapped conversation.



**cvReturnCtl** (supplied) specifies when the local LU is to return control to the transaction program and what type of session allocation is to be used. If the local LU fails to obtain a session for the basic conversation, an allocation error is reported either on this routine or on a subsequent routine. If the remote LU rejects the allocation request, an allocation error is reported on a subsequent routine. The following values are defined:

**kWhenAllocReturn** allocates a session before returning control to the local transaction program. A session-allocation error is reported upon return from the **BCAllocate** routine.

**kDelayAllocReturn** allocates a session after returning control to the local transaction program. A session-allocation error is reported upon return from a subsequent **MacAPPC** routine.

**kImmedAllocReturn** allocates a session only if a session is immediately available and returns control to the local transaction program. A session is immediately available when it is a free first-speaker session. A session-allocation error is reported upon return from the **BCAllocate** routine if a session is not immediately available.

**cvSyncType** (supplied) specifies the synchronization level that the local and remote transaction programs can use for the conversation. The values are defined as follows:

**kNoSync** specifies that the transaction programs do not perform confirmation processing nor sync-point processing on this conversation. The transaction programs do not execute any routines and do not recognize any returned parameters relating to confirmation or synchronization functions.

**kConfirmSync** specifies that transaction programs can perform confirmation processing but not sync-point processing on this conversation. The transaction programs do not execute any routines and do not recognize any returned parameters relating to the synchronization functions.

**kSyncPtSync** specifies that transaction programs can perform both confirmation processing and sync-point processing on this conversation.

❖ *Note:* At the time of publication, sync-point services were not supported.

**cvPIPUsed** (supplied) specifies whether or not program initialization parameters (PIPs) are to be sent to the remote transaction program. A value of **TRUE** specifies that PIP data is present; **FALSE** specifies that PIP data is not present.

**cvPIPPtr** (supplied) specifies an array of pointers to program initialization parameters. The last pointer must be followed by one that is **NIL**. The maximum number of parameters is defined by the value of the **kMaxPIP** constant, with a total space limitation specified by the **cvPIPBuffSize** parameter (see **cvPIPBuffSize** for more information about space limitations). This array is ignored if the **cvPIPUsed** parameter is set to **FALSE**.

**cvPIPSize** (supplied) specifies an array of sizes that specifies the size for each PIP in the **cvPIPPtr** array. The last size must be followed by a size of 0.

**cvSecType** (supplied/returned) specifies the type of access-security information that is to be used by the remote LU to validate access to the remote transaction program and its resources. The following values are defined:

**kNoSec** specifies that access-security information is not to be used.



`kSameSec` specifies that the security information to be used is from the local transaction program executing the `BCAllocate` request, so that the security level remains the same as set by the previous allocation request. The allocation request carries the user name of the local transaction program and is indicated as already verified (that is, no password is sent). If the local transaction program was not previously allocated, the `cvSecType` parameter returns the `kNoSec` constant.

`kProgSec` specifies that the security information to be used is contained in the `cvUserName`, `cvUserPswd`, and optionally the `cvUserProf` parameters.

`cvConvID` (returned) indicates the conversation ID of the allocated conversation.

## Notes

Successful completion of the `BCAllocate` routine does not indicate successful allocation. This routine can only return session-allocation errors (with the `cvReturnCtl` parameter set to the `kWhenSessAllocReturn` or `kImmediateReturn` constant). All other allocation errors are reported on subsequent routines.

When control returns and no error was encountered, the conversation is in send state.

For IBM equipment, make sure that the PIP data is in the format that the receiving transaction program expects. For example, you may need to execute the `TPAsciiToEbcdic` routine.

<b>Result code</b>	<code>appcNoErr</code>	Routine succeeded
	<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
	<code>appcExec</code>	Routine executing; asynchronous request not complete

**See also** `MCAAllocate`, `BCDeallocate`



---

## BCConfirm

### Summary

The `BCConfirm` routine flushes the send buffer, transmits a request for confirmation to the remote transaction program, and waits for a reply. The remote transaction program replies with either a confirmation or an error. This routine allows the local and remote transaction programs to synchronize their processing. This routine is not available on conversations allocated with the synchronization level of none.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0082	byte	←	cvReqToSendRcvd

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvReqToSendRcvd** (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
appcExec	Routine executing; asynchronous request not complete

### See also

`MCConfirm`, `BCConfirmed`



---

## BCConfirmed

<b>Summary</b>	The BCConfirmed routine sends a confirmation reply to the remote transaction program when a confirmation request is received.			
<b>Parameters</b>	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0020	byte	←	appcConvState
	0026	long	→	cvTPCBPtr
	002A	long	→	cvCVCBPtr
<b>Description</b>	<b>cvTPCBPtr</b> (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).			
	<b>cvCVCBPtr</b> (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).			
<b>Notes</b>	When control returns and no error is encountered, the conversation state changes as follows: if the conversation was in confirm state, it goes to receive state; if the conversation was in confirm/send state, it goes to send state; if the conversation was in confirm/deallocate state, it goes to deallocate state.			
<b>Result code</b>	appcNoErr	Routine succeeded		
	appcFail	Routine failed; look in appcHiResult and appcLoResult		
	appcExec	Routine executing; asynchronous request not complete		
<b>See also</b>	MCConfirmed, BCConfirm			



---

## BCDeallocate

### Summary

The `BCDeallocate` routine flushes the send buffer and deallocates the conversation from the transaction program. It can also include the function of the `BCCconfirm` routine. The conversation becomes unassigned when deallocation is complete.

---

### Important

Your transaction program must execute a `BCDeallocate` routine to end a basic conversation. After the `BCDeallocate` routine has been executed, no more basic conversation routines can be executed for that deallocated basic conversation.

---

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0085	byte	→	cvDeallocType
0070	long	→	cvDataPtr
0074	word	→	cvDataSize

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB) to be deallocated.

**cvDeallocType** (supplied) specifies the type of deallocation:

**kSyncDealloc** specifies that MacAPPC should perform deallocation based on the synchronization level allocated to this conversation. A synchronization level of none will perform a deallocation as if the **kFlushDealloc** constant had been specified. A synchronization level of confirm will perform a deallocation as if the **kConfirmDealloc** constant had been specified.

**kFlushDealloc** specifies that MacAPPC should execute the function of **MCFlush** and deallocate the conversation normally.

**kConfirmDealloc** specifies that MacAPPC should execute the function of the **BCCconfirm** routine and, if it is successful, deallocate the conversation normally; if it is not successful, the state of the conversation is determined by the result code.

**kAbendProgDealloc** specifies that MacAPPC should execute the function of the **BCFlush** routine when the conversation is in send or defer state, and deallocate the conversation normally. If the conversation is in receive state, data can be lost. **kAbendProgDealloc** is intended to be used by a transaction program when it detects an error condition that prevents completion of the transaction.



`kAbendSvcDealloc` is intended to be used by a service transaction program when it detects an error condition that prevents completion of the transaction. Its specific use and meaning are defined by the service transaction program.

`kAbendTimerDealloc` is intended to be used by an LU services component, such as one that processes mapped conversation routines, when it detects an error condition caused by its peer LU services component in the remote LU. An example is a format error in control information sent by the peer LU services component. The specific use and meaning are defined by the transaction program.

`kLocalDealloc` specifies that MacAPPC should deallocate the conversation locally. The transaction program should specify this type of deallocation if, and only if, the conversation is in deallocate state.

`cvDataPtr` (supplied) specifies a pointer to error data to be written to the local and remote LU error logs. Log data is not accepted if error logging support is not configured for the local and remote LUs. The size of the buffer is specified by the `cvDataSize` parameter. If log data is not specified, this pointer must be NIL.

`cvDataSize` (supplied) specifies the size of the log data buffer pointed to by the `cvDataPtr` parameter. The maximum size of the buffer is specified by the `kMaxLogData` constant.

## Notes

When control returns, if no errors were encountered, the conversation enters reset state. When the `cvDeallocType` parameter is set to the `kSyncDealloc` constant and the partner sends an error response, the conversation enters receive state.

## Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

## See also

`MCDeallocate`, `BCAllocate`



---

## BCFlush

### Summary

The `BCFlush` routine sends the information that is in the LU's send buffer to the remote transaction program. Information is buffered in the send buffer by the `BCAllocate`, `BCDeallocate`, `BCSendData`, and `BCSendError` routines.

`BCFlush` is useful for optimization of processing between the local and remote transaction programs. The LU normally buffers the data records from consecutive `BCSendData` routines until it has a sufficient amount for transmission. At that time it transmits the buffered data records. However, the local transaction program can execute `BCFlush` in order to cause the LU to transmit the buffered data records. In this way, the local transaction program can minimize the delay in the remote transaction program's processing of the data records.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

### Notes

When the routine completes without error, no state change occurs. If an error is detected, the conversation enters either receive state or deallocate state, depending on the error.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`MCFlush`



---

## BCGetAttributes

### Summary

The `BCGetAttributes` routine returns information about the specified conversation.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
004A	long	⇒	cvFullLclLUName
0042	long	⇒	cvRmtLUName
0046	long	⇒	cvFullRmtLUName
004E	long	⇒	cvModeName
0056	long	⇒	cvUserName
005E	long	⇒	cvUserProf
0062	long	⇒	cvLUWName
0066	long	⇒	cvLUWID
006A	long	⇒	cvLUWCorr
0089	byte	←	cvSyncType
006E	word	←	cvLUWSeq
003A	long	←	cvConvID
003E	long	←	cvProgID

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvFullLclLUName** (supplied/modified) specifies a pointer to space where the fully qualified network name of the local LU can be returned. The space must be at least two times the value of the `kMaxName` constant plus 1 byte. The name is returned as two strings concatenated by a period: `NETID.NETNAME`, where `NETID` is the network ID and `NETNAME` is the network LU name. When there is no network ID, only `NETNAME` is returned. If the pointer is `NIL`, the name is not returned.

**cvRmtLUName** (supplied/modified) specifies a pointer to space where the local name of the remote LU—that is, where the remote transaction program is located—can be returned. The space must be at least as large as the value of the `kMaxName` constant plus 1 byte. This is the name by which the local LU knows the remote LU for the purpose of allocating a conversation. If the pointer is `NIL`, the name is not returned.

**cvFullRmtLUName** (supplied/modified) specifies a pointer to space where the fully qualified network name for the remote LU can be returned. The space must be at least two times the value of the `kMaxName` constant plus 1 byte. The name has the same form as that specified in the `cvFullLclLUName` parameter. If the remote LU's fully qualified name is not known, a `NULL` string is returned. If the pointer is `NIL`, the name is not returned.



**cvModeName** (supplied/modified) specifies a pointer to space where the mode name for the session allocated to the conversation can be returned. The space must be at least as large as the value of the `kMaxName` constant plus 1 byte. If the pointer is NIL, the name is not returned.

**cvUserName** (supplied/modified) specifies a pointer to space where the user ID can be returned. The space must be at least as large as the value of the `kMaxSecName` constant plus 1 byte. The remote LU uses this value and the password to verify the identity of the transaction program making the allocation request. In addition, the remote LU can use the value of the `cvUserName` parameter for auditing or accounting purposes, or it can use the value of `cvUserName`, together with the profile (see `cvUserProf`), to determine which remote transaction programs the local transaction program can access and which resources the remote transaction program can access. If the pointer is NIL, the user ID is not returned.

**cvUserProf** (supplied/modified) specifies a pointer to space where a profile to be used in place of or in addition to the user ID specified in the `cvUserName` parameter can be returned. The space must be at least as large as the value of the `kMaxSecName` constant plus 1 byte. The remote LU can use this value, in addition to or in place of the `cvUserName` parameter, to determine which remote transaction programs the local transaction program can access, and which resources the remote transaction program can access. If the pointer is NIL, the profile is not returned.

**cvLUWName** (supplied/modified) specifies a pointer to space where the LU name portion of the logical-unit-of-work (LUW) can be returned. The space must be at least as large as the value of the `kMaxLUWName` constant plus 1 byte. The LUW identifier is created and maintained by the LU, which uses it for accounting purposes. If the pointer is NIL, the name is not returned.

**cvLUWID** (supplied/modified) specifies a pointer to space where the unique ID portion of the logical-unit-of-work can be returned. The space must be at least as long as the value of the `kMaxLUWID` constant plus 1 byte. This is not a printable string. If the pointer is NIL, the LUW ID is not returned.

**cvLUWCorr** (supplied/modified) specifies a pointer to space where the conversation correlator can be returned. The space must be at least as long as the value of the `kMaxLUWCorr` constant plus 1 byte. The conversation correlator is created and maintained by the LU. This is not a printable string. If the pointer is NIL, the name is not returned.

**cvSyncType** (returned) indicates the synchronization level that the local and remote transaction programs can use for the conversation. The following values are defined:

`kNoSync` indicates that the transaction programs do not perform confirmation processing on this conversation. The transaction programs do not execute any routines and do not recognize any returned parameters relating to these synchronization functions.

`kConfirmSync` indicates that transaction programs can perform confirmation processing but not sync-point processing on this conversation. The transaction programs do not execute any routines and do not recognize any returned parameters relating to the synchronization functions.

`kSyncPtSync` indicates that transaction programs can perform both confirmation processing and sync-point processing on this conversation.

❖ *Note:* At the time of publication, sync-point services were not supported.



**cvLWSeq** (returned) is the logical-unit-of-work sequence number assigned to this conversation.

**cvConvID** (returned) returns the conversation ID.

**cvProgID** (returned) returns the program ID.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **MCGetAttributes**



---

## BCPostOnReceipt

### Summary

The `BCPostOnReceipt` routine requests a conversation to be posted when information is available for the transaction program to receive. Execute a `BCTest` routine after `BCPostOnReceipt` to determine if posting has occurred. Execute a `CVWait` routine after `BCPostOnReceipt` to wait for posting to occur.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0088	byte	→	cvFillType
0074	word	→	cvDataSize

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvFillType** (supplied) specifies whether or not posting for data is to occur for a logical length record:

**kBufferFill** specifies that the conversation is to be posted when the specified amount arrives, regardless of the logical record format.

**kLLFill** specifies that the conversation is to be posted when a complete or truncated logical length record arrives, or when at least the specified amount of data arrives for the logical length record.

**cvDataSize** (supplied) specifies the minimum amount of data that will cause posting to occur. This is used with the `cvFillType` parameter to determine when to post the conversation. The `cvDataSize` parameter must be small enough so that the amount of data that will cause posting does not exceed the size of the LU's receive buffer. If that buffer is exceeded, posting will never occur. The proper size for the LU's receive buffer depends upon several factors; you will have to empirically test different values for `cvDataSize` to find the correct value.

❖ *Note:* If a small value has been specified for `cvDataSize` and a large data block has begun to arrive, you can subsequently use either a `BCReceiveAndWait` routine to receive the entire data block or a `BCReceiveImmediate` routine to receive only the data that has been received by the LU.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`MCPPostOnReceipt`, `BCTest`, `CVWait`



---

## BCPrepareToReceive

### Summary

The `BCPrepareToReceive` routine flushes the send buffer and changes the conversation from send state to receive state. This routine can also include the function of the `BCConfirm` routine, requesting confirmation before entering receive state.

### Parameters

000C	long	→	<code>ioCompletion</code>
0018	word	→	<code>appcRefNum</code>
001A	word	→	<code>appcOpCode</code>
0022	long	→	<code>appcUserRef</code>
0010	word	←	<code>ioResult</code>
001C	word	←	<code>appcHiResult</code>
001E	word	←	<code>appcLoResult</code>
0020	byte	←	<code>appcConvState</code>
0026	long	→	<code>cvTPCBPtr</code>
002A	long	→	<code>cvCVCBPtr</code>
0086	byte	→	<code>cvPrepToRcvType</code>
0087	byte	→	<code>cvLockType</code>

### Description

**`cvTPCBPtr`** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**`cvCVCBPtr`** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**`cvPrepToRcvType`** (supplied) specifies the type of request:

`kFlushRcv` indicates that the send buffer should be flushed (as in the `BCFlush` routine) and the conversation should enter receive state.

`kConfirmRcv` indicates that the data in the buffer should be sent and that confirmation will be requested before entering receive state.

**`cvLockType`** (supplied) specifies when control is to be returned to the local transaction program if the function of `BCConfirm` is performed. This parameter is ignored if the conversation does not support confirmation processing.

`kShortLock` specifies that control should be returned to the local transaction program when an affirmative reply is received.

`kLongLock` specifies that control should be returned to the local transaction program when data following confirmation is received.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`MCPPrepareToReceive`



---

## BCReceiveAndWait

### Summary

The `BCReceiveAndWait` routine receives information that has arrived for the basic conversation. The routine can wait for more information to arrive if the request is not yet satisfied. The information can be data, conversation status, or a request for confirmation. The transaction program can execute this routine when the conversation is in send state, which flushes the buffer and places the conversation in receive state.

The transaction program receives only one kind of information at a time; that is, it can receive data or control information, but it cannot receive both simultaneously.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0088	byte	→	cvFillType
0070	long	⇒	cvDataPtr
0074	word	↔	cvDataSize
0084	byte	←	cvWhatRcvd
0082	byte	←	cvReqToSendRcvd

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvFillType** (supplied) specifies whether or not data is to be received according to the logical record format.

`kBufferFill` receives the specified amount of data independent of its logical record format.

`kLLFill` receives one complete or truncated logical record, or up to the specified amount of the logical record.

**cvDataPtr** (supplied/modified) specifies a pointer to space into which received data is copied. The size of the space is specified in the `cvDataSize` parameter.

**cvDataSize** (supplied/returned) specifies the maximum amount of data the transaction program is to receive. When control returns to the transaction program, this field contains the length of data received.

**cvWhatRcvd** (returned) indicates what was received, as follows:

`kDataRcvd` indicates that data was received. It is returned only when the `cvFillType` parameter is set to the `kBufferFill` constant.

`kDataComplRcvd` indicates that a complete logical length record (LL) or the end of a previously incomplete LL was received. It is returned only when the `cvFillType` parameter is set to the `kLLFill` constant.



kDataIncomplRcvd indicates that an incomplete LL was received. It is returned only when the cvFillType parameter is set to the kLLFill constant.

kLLTruncRcvd indicates that a complete logical length record was not received, and the rest of the data was discarded. No data is returned. It is returned only when the cvFillType parameter is set to the kLLFill constant.

kSendRcvd indicates that the remote transaction program has entered receive state and sent a send indication. No data is returned. The conversation enters send state. The local transaction program can now execute BCSendData.

kConfirmRcvd indicates that the remote transaction program has requested confirmation. No data is returned. The conversation enters confirm state.

kConfirmSendRcvd indicates that the remote transaction program has executed a prepare-to-receive confirm. The conversation enters confirm/send state.

kConfirmDeallocRcvd indicates that the remote transaction program has executed a deallocate confirm. The conversation enters confirm/deallocate state.

**cvReqToSendRcvd** (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

## Notes

When control returns and no errors are encountered, the conversation is in receive state unless otherwise indicated by the value of the cvWhatRcvd parameter.

For IBM equipment, be aware of the format that the sending transaction program is using. For example, you may need to execute the TPEbcdicToAscii routine.

## Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

## See also

MReceiveAndWait, BReceiveImmediate



---

## BCReceiveImmediate

### Summary

The `BCReceiveImmediate` routine receives information that is available for the conversation. The information can be data, conversation status, or a request for confirmation. Control is returned to the transaction program with an indication of what kind of information was received, if any.

The transaction program receives only one kind of information at a time; that is, it can receive data or control information, but it cannot receive both simultaneously.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0088	byte	→	cvFillType
0070	long	⇒	cvDataPtr
0074	word	↔	cvDataSize
0084	byte	←	cvWhatRcvd
0082	byte	←	cvReqToSendRcvd

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvFillType** (supplied) specifies whether or not data is to be received according to the logical record format.

`kBufferFill` receives the specified amount of data independent of its logical length record format.

`kLLFill` receives one complete or truncated logical record, or up to the specified amount of the logical length record.

**cvDataPtr** (supplied/modified) specifies a pointer to space into which received data is copied. The size of the space is specified in the `cvDataSize` parameter.

**cvDataSize** (supplied/returned) specifies the maximum amount of data the transaction program is to receive. When control returns to the transaction program, this field contains the length of data received.

**cvWhatRcvd** (returned) indicates what was received, as follows:

`kDataRcvd` indicates that data was received. It is returned only when the `cvFillType` parameter is set to the `kBufferFill` constant.

`kDataComplRcvd` indicates that a complete logical length record (LL) or the end of a previously incomplete LL was received. It is returned only when the `cvFillType` parameter is set to the `kLLFill` constant.



kDataIncomplRcvd indicates that an incomplete LL was returned. It is returned only when the cvFillType parameter is set to the kLLFill constant.

kLLTruncRcvd indicates that a complete logical length record was not received, and the rest of the data was discarded. No data is returned. It is returned only when the cvFillType parameter is set to the kLLFill constant.

kSendRcvd indicates that the remote transaction program has entered receive state and sent a send indication. No data is returned. The conversation enters send state. The local transaction program can now execute BCSendData.

kConfirmRcvd indicates that the remote transaction program has requested confirmation. No data is returned. The conversation enters confirm state.

kConfirmSendRcvd indicates that the remote transaction program has executed a prepare-to-receive confirm. The conversation enters confirm/send state.

kConfirmDeallocRcvd indicates that the remote transaction program has executed a deallocate confirm. The conversation enters confirm/deallocate state.

**cvReqToSendRcvd** (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

#### Notes

When control returns and no errors are encountered, the conversation is in receive state unless otherwise indicated by the cvWhatRcvd parameter.

For IBM equipment, be aware of the format that the sending TP is using. For example, you may need to execute the TPEbcdicToAscii routine.

#### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

#### See also

MCRceiveImmediate, BCReceiveAndWait



---

## BCRequestToSend

### Summary

The `BCRequestToSend` routine is used by the local transaction program to request to enter send state. A request-to-send indicator is sent to the remote transaction program which can either honor or ignore the request. If the request is honored, the conversation enters send state for the local transaction program when a send indication is received from the remote transaction program.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

### Notes

No state change occurs.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`MCRequestToSend`



---

## BCSendData

### Summary

The `BCSendData` routine transmits data to the remote transaction program. The data must be formatted into logical records by the local transaction program, but it is sent independently of the logical record format.

The data is buffered in the local LU's send buffer and is not transmitted to the remote transaction program until the send capacity is exceeded. Transmission of the contents of the send buffer is forced by the `BCFlush` routine or by routines that include the flush function (for example, `BCPrepareToReceive`).

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0070	long	→	cvDataPtr
0074	word	→	cvDataSize
0082	byte	←	cvReqToSendRcvd

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvDataPtr** (supplied) points to the data to be sent. The size of the data is specified by the `cvDataSize` parameter. This data must be formatted into logical records by the local transaction program before it is sent. Each logical record consists of a 2-byte length field followed by the data. The 2-byte length field contains the 15-bit length of the record and a high-order bit that is ignored (this bit is used by mapped conversation routines). The length of the record includes the length of the 2-byte length field. For example, a 40-byte record would be sent in a 42-byte logical record with a length field of 42. Logical record lengths of \$0000, \$0001, \$8000, and \$8001 are invalid.

**cvDataSize** (supplied) contains the length of the data to be sent. This parameter is independent of the logical record format.

**cvReqToSendRcvd** (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

### Notes

When no error is encountered, the conversation remains in send state.

For IBM equipment, be aware of the format that the remote transaction program is expecting. For example, you may need to execute the `TPAsciiToEbcdic` routine.



<b>Result code</b>	appcNoErr	Routine succeeded
	appcFail	Routine failed; look in appcHiResult and appcLoResult
	appcExec	Routine executing; asynchronous request not complete
<b>See also</b>	MCSendData	



---

## BCSendError

### Summary

The `BCSendError` routine sends an error notification to the remote transaction program. If the conversation is in send state, the send buffer is flushed. When the routine is completed with no errors, the conversation is in send state and the remote transaction program is in receive state.

### Parameters

000C	long	→	<code>ioCompletion</code>
0018	word	→	<code>appcRefNum</code>
001A	word	→	<code>appcOpCode</code>
0022	long	→	<code>appcUserRef</code>
0010	word	←	<code>ioResult</code>
001C	word	←	<code>appcHiResult</code>
001E	word	←	<code>appcLoResult</code>
0020	byte	←	<code>appcConvState</code>
0026	long	→	<code>cvTPCBPtr</code>
002A	long	→	<code>cvCVCBPtr</code>
008D	byte	→	<code>cvErrorType</code>
007E	long	→	<code>cvSenseData</code>
0070	long	→	<code>cvDataPtr</code>
0074	word	→	<code>cvDataSize</code>
0082	byte	←	<code>cvReqToSendRcvd</code>

### Description

**`cvTPCBPtr`** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**`cvCVCBPtr`** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**`cvErrorType`** (supplied) specifies the type of error, as follows:

`kSvcError` indicates a service transaction program error.

`kProgError` indicates a transaction program error.

**`cvSenseData`** (supplied) is reserved for use by the MacAPPC drivers.

**`cvDataPtr`** (supplied) specifies a pointer to error data to be written to the local and remote LU error logs. Log data is not accepted if error logging support is not configured for the local and remote LUs. The size of the buffer is specified by the `cvDataSize` parameter. If log data is not specified, this pointer must be NIL.

**`cvDataSize`** (supplied) specifies the size of the log data buffer pointed to by the `cvDataPtr` parameter. The maximum size of the buffer is specified by the `kMaxLogData` constant.

**`cvReqToSendRcvd`** (returned) returns TRUE if the remote transaction program has issued a request-to-send, thus requesting that the local transaction program enter receive state and place the remote transaction program in send state.

### Notes

If no errors are encountered, the program enters send state.

If a race condition arises where both the local and remote transaction programs issue error indications, the program that was in receive state wins the race and the program that was in send state receives a program-error indication.



<b>Result code</b>	appcNoErr	Routine succeeded
	appcFail	Routine failed; look in appcHiResult and appcLoResult
	appcExec	Routine executing; asynchronous request not complete
<b>See also</b>	MCSendError	



---

## BCTest

### Summary

The **BCTest** routine tests the specified conversation to see whether posting has occurred or whether a request-to-send notification has been received. If a request-to-send has been received, the routine returns **appcNoErr**. If posting has occurred, the routine returns **appcNoErr** and **appcLoResult** is set to one of the following constants:

**dataAvail** indicates that data has arrived.

**ctlAvail** indicates that control information has arrived.

When testing for posting, posting must be previously activated for a conversation with the **BCPostOnReceipt** routine.

### Parameters

000C	long	→	<b>ioCompletion</b>
0018	word	→	<b>appcRefNum</b>
001A	word	→	<b>appcOpCode</b>
0022	long	→	<b>appcUserRef</b>
0010	word	←	<b>ioResult</b>
001C	word	←	<b>appcHiResult</b>
001E	word	←	<b>appcLoResult</b>
0020	byte	←	<b>appcConvState</b>
0026	long	→	<b>cvTPCBPtr</b>
002A	long	→	<b>cvCVCBPtr</b>
008E	byte	→	<b>cvTestType</b>

### Description

**cvTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**cvCVCBPtr** (supplied) specifies a pointer to an existing Conversation Control Block (CVCB).

**cvTestType** (supplied) specifies the condition to be tested, as follows:

**kPostTest** specifies to test whether the conversation has been posted.

**kReqToSendTest** specifies to test whether request-to-send notification has been received from the remote transaction program.

### Result code

<b>appcNoErr</b>	Routine succeeded
<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
<b>appcExec</b>	Routine executing; asynchronous request not complete

### See also

**MCTest**, **BCPostOnReceipt**



---

---

## Summary of the MacAPPC Conversation Driver

This section provides a summary of the constants, data structures, and routines for use with the MacAPPC Conversation Driver.

---

### Constants

The following constants are available for use with the MacAPPC Conversation Driver.

{ cvWhatRcvd values }

```
kNullRcvd =          0;
kDataRcvd =          1;
kDataComplRcvd =      2;
kDataIncomplRcvd =    3;
kLLTruncRcvd =        4;
kSendRcvd =           5;
kConfirmRcvd =         6;
kConfirmSendRcvd =    7;
kConfirmDeallocRcvd = 8;
kDataTruncRcvd =       9;
kFMHDataComplRcvd =   10;
kFMHDataIncomplRcvd = 11;
kFMHDataTruncRcvd =   12;
kTakeSyncPtRcvd =     13; { not supported }
kTakeSyncPtSendRcvd = 14; { not supported }
kTakeSyncPtDeallocRcvd = 15; { not supported }
```

{ cvDeallocType values }

```
kSyncDealloc =        1;
kFlushDealloc =        2;
kAbendProgDealloc =    3;
kAbendSvcDealloc =      4;
kAbendTimerDealloc =   5;
kLocalDealloc =         6;
kConfirmDealloc =       7;
kAbendDealloc =         8;
```

{ cvPrepToRcvType values }

```
kFlushRcv =           0;
kConfirmRcv =          1;
kSyncLevelRcv =        2;
```

{ cvLockType values }

```
kShortLock =          0;
kLongLock =            1;
```



```

{ cvFillType values }

kBufferFill =      0;
kLLFill =          1;

{ cvSyncType values }

kNoSync =          0;
kConfirmSync =     1;
kSyncPtSync =      2;          { not supported }

{ cvReturnCtl values }

kWhenAllocReturn = 0;
kDelayAllocReturn = 1;
kImmedAllocReturn = 2;

{ cvSecType values }

kNoSec =           0;
kSameSec =         1;
kProgSec =         2;

{ cvConvType values }

kBasicConv =       0;
kMappedConv =      1;

{ cvErrorType values }

kSvcError =        0;
kProgError =       1;
kAllocError =      2;          { reserved }

{ cvTestType values }

kPostTest =        0;
kReqToSendTest =   1;

{ mcpbMapCmd values }

kSendMapping =     0;
kRcvMapping =      1;

{ mcpbResult values }

mcNoErr =          0;
mcErr =            1;
mcMapNameErr =     2;
mcDupMapNameErr =  3;

{ mcpbRcvMode values }

kTruncMode =       0;
kIncomplMode =     1;

```



---

## Data types

The following data types are available for use with the MacAPPC Conversation Driver.

cvParam:

```
(
    cvTPCBPtr      : Ptr;          { TPCB pointer }
    cvCVCBPtr      : Ptr;          { CVCB pointer }
    cvPIPBufPtr    : Ptr;          { PIP buffer pointer }
    cvPIPBufSize   : INTEGER;      { PIP buffer size }
    cvMapBufPtr    : Ptr;          { mapped conversation buffer pointer }
    cvMapBufSize   : INTEGER;      { mapped conversation buffer size }
    cvConvID       : LONGINT;      { conversation ID }
    cvProgID       : LONGINT;      { transaction program ID }
    cvRmtLUName    : StringPtr;    { remote LU name pointer }
    cvFullRmtLUName : StringPtr;    { fully qualified RLU name pointer }
    cvFullLclLUName : StringPtr;    { fully qualified LLU name pointer }
    cvModeName     : StringPtr;    { mode name pointer }
    cvRmtProgName  : StringPtr;    { remote program name pointer }
    cvUserName     : StringPtr;    { user name pointer }
    cvUserPswd     : StringPtr;    { user password pointer }
    cvUserProf     : StringPtr;    { user profile pointer }
    cvLUWName      : StringPtr;    { Logical Unit of Work LU name pointer }
    cvLUWID        : StringPtr;    { LUW identifier pointer }
    cvLUWCorr      : StringPtr;    { LUW conversation correlator pointer }
    cvLUWSeq       : INTEGER;      { LUW sequence number }
    cvDataPtr      : Ptr;          { data buffer pointer }
    cvDataSize     : INTEGER;      { data buffer size }
    cvMapName      : StringPtr;    { map name pointer }
    cvMapProc      : ProcPtr;      { mapping procedure pointer }
    cvSenseData    : LONGINT;      { reserved }
    cvReqToSendRcvd : Byte;        { request to send received }
    cvFMHdrs       : Byte;        { FM headers in data record }
    cvWhatRcvd     : Byte;        { what was received }
    cvDeallocType  : Byte;        { deallocation type }
    cvPrepToRcvType : Byte;        { prepare to receive type }
    cvLockType     : Byte;        { prepare to receive lock }
    cvFillType     : Byte;        { logical record receive }
    cvSyncType     : Byte;        { synchronization level }
    cvReturnCtl    : Byte;        { allocate return control }
    cvSecType      : Byte;        { security type }
    cvConvType     : Byte;        { conversation type }
    cvErrorType    : Byte;        { send error type }
    cvTestType     : Byte;        { test type }
    cvPIPUsed      : Byte;        { program parameters used }
    cvConvIDCount  : INTEGER;      { # of conversation IDs }
    cvConvIDList   : ARRAY[1..kMaxConvID] OF LONGINT; { list of conv IDs }
    cvPIPPtr       : ARRAY[1..kMaxPIP] OF Ptr; { array of PIP ptrs }
    cvPIPSize      : ARRAY[1..kMaxPIP] OF INTEGER; { array of PIP sizes }
);
```



## Mapping parameter block

```
APPCMCPB =                RECORD
    mcpbMapCmd              : SignedByte;    { MC request }
    mcpbResult              : INTEGER;        { mapper return code }
    mcpbMapName             : StringPtr;      { map name pointer }
    mcpbDataPtr             : Ptr;            { data pointer }
    mcpbDataSize            : INTEGER;        { data length }
    mcpbBuffPtr             : Ptr;            { buffer pointer }
    mcpbBuffSize            : INTEGER;        { buffer length }
    mcpbTransMapName        : Boolean;        { map name translation required }
    mcpbFMHdrs              : Boolean;        { FMH data contains FM headers }
    mcpbRcvMode             : SignedByte;    { receive mode }
    mcpbAPPCMCPBPtr        : Ptr;            { APPC parameter block pointer }
END;

APPCMCPBPtr =              ^APPCMCPB;
```



---

## Mapped routines

The following mapped conversation routines are available for use with the MacAPPC Conversation Driver.

### MCAAllocate

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0034	long	→	cvMapBuffPtr
0038	word	→	cvMapBuffSize
002E	long	→	cvPIPBuffPtr
0032	word	→	cvPIPBuffSize
0042	long	→	cvRmtLUName
0052	long	→	cvRmtProgName
004E	long	→	cvModeName
0056	long	→	cvUserName
005A	long	→	cvUserPswd
005E	long	→	cvUserProf
007A	long	→	cvMapProc
008A	byte	→	cvReturnCtl
0089	byte	→	cvSyncType
008F	byte	→	cvPIPUsed
0492	----	→	cvPIPPtr[]
0892	----	→	cvPIPSize[]
008B	byte	↔	cvSecType
003A	long	←	cvConvID

### MCCConfirm

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0082	byte	←	cvReqToSendRcvd



## MCConfirmed

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

## MCDeallocate

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0085	byte	→	cvDeallocType

## MCFlush

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

## MCGetAttributes

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
004A	long	⇒	cvFullLclLUName
0042	long	⇒	cvRmtLUName
0046	long	⇒	cvFullRmtLUName
004E	long	⇒	cvModeName



0056	long	⇒	cvUserName
005E	long	⇒	cvUserProf
0062	long	⇒	cvLUWName
0066	long	⇒	cvLUWID
006A	long	⇒	cvLUWCorr
0089	byte	←	cvSyncType
006E	word	←	cvLUWSeq
003A	long	←	cvConvID
003E	long	←	cvProgID

#### **MCPPostOnReceipt**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0074	word	→	cvDataSize

#### **MCPPrepareToReceive**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0086	byte	→	cvPrepToRcvType
0087	byte	→	cvLockType

#### **MCRReceiveAndWait**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0070	long	⇒	cvDataPtr
0074	word	↔	cvDataSize
0076	long	⇒	cvMapName
0084	byte	←	cvWhatRcvd
0082	byte	←	cvReqToSendRcvd



### **MReceiveImmediate**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0070	long	⇒	cvDataPtr
0074	word	↔	cvDataSize
0076	long	⇒	cvMapName
0084	byte	←	cvWhatRcvd
0082	byte	←	cvReqToSendRcvd

### **MRequestToSend**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

### **MCSendData**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0070	long	→	cvDataPtr
0074	word	→	cvDataSize
0076	long	→	cvMapName
0083	byte	→	cvFMHdrs
0082	byte	←	cvReqToSendRcvd

### **MCSendError**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult



001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0082	byte	←	cvReqToSendRcvd

## MCTest

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
008E	byte	→	cvTestType



---

## Type-independent routines

The following type-independent routines are available for use with the MacAPPC Conversation Driver.

### CVBackout

At the time of publication, CVBackout was not supported.

### CVGetType

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
008C	byte	←	cvConvType

### CVSyncPoint

At the time of publication, CVSyncPoint was not supported.

### CVWait

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0092	----	→	cvCVCBList[]
0090	word	↔	cvCVCBIndex



---

## Basic conversation routines

The following basic conversation routines are available for use with the MacAPPC Conversation Driver.

### BCAllocate

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
002E	long	→	cvPIPBuffPtr
0032	word	→	cvPIPBuffSize
0042	long	→	cvRmtLUName
0052	long	→	cvRmtProgName
004E	long	→	cvModeName
0056	long	→	cvUserName
005A	long	→	cvUserPswd
005E	long	→	cvUserProf
008C	byte	→	cvConvType
008A	byte	→	cvReturnCtl
0089	byte	→	cvSyncType
008F	byte	→	cvPIPUsed
0492	----	→	cvPIPPtr[]
0892	----	→	cvPIPSize[]
008B	byte	↔	cvSecType
003A	long	←	cvConvID

### BCCConfirm

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0082	byte	←	cvReqToSendRcvd

### BCCConfirmed

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef



0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

#### BCDeallocate

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0085	byte	→	cvDeallocType
0070	long	→	cvDataPtr
0074	word	→	cvDataSize

#### BCFlush

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

#### BCGetAttributes

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
004A	long	⇒	cvFullLclLUName
0042	long	⇒	cvRmtLUName
0046	long	⇒	cvFullRmtLUName
004E	long	⇒	cvModeName
0056	long	⇒	cvUserName
005E	long	⇒	cvUserProf
0062	long	⇒	cvLUWName



0066	long	⇒	cvLUWID
006A	long	⇒	cvLUWCorr
0089	byte	←	cvSyncType
006E	word	←	cvLUWSeq
003A	long	←	cvConvID
003E	long	←	cvProgID

#### **BCPostOnReceipt**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0088	byte	→	cvFillType
0074	word	→	cvDataSize

#### **BCPrepareToReceive**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0086	byte	→	cvPrepToRcvType
0087	byte	→	cvLockType

#### **BCReceiveAndWait**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0088	byte	→	cvFillType
0070	long	⇒	cvDataPtr
0074	word	↔	cvDataSize
0084	byte	←	cvWhatRcvd
0082	byte	←	cvReqToSendRcvd



### BCReceiveImmediate

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0088	byte	→	cvFillType
0070	long	⇒	cvDataPtr
0074	word	↔	cvDataSize
0084	byte	←	cvWhatRcvd
0082	byte	←	cvReqToSendRcvd

### BCRequestToSend

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr

### BCSendData

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
0070	long	→	cvDataPtr
0074	word	→	cvDataSize
0082	byte	←	cvReqToSendRcvd

### BCSendError

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult



0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
008D	byte	→	cvErrorType
007E	long	→	cvSenseData
0070	long	→	cvDataPtr
0074	word	→	cvDataSize
0082	byte	←	cvReqToSendRcvd

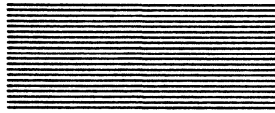
#### BCTest

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	cvTPCBPtr
002A	long	→	cvCVCBPtr
008E	byte	→	cvTestType

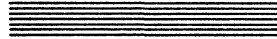








## **Chapter 5**



# **MacAPPC Control Operator Driver**



This chapter describes the MacAPPC Control Operator Driver (.CO62), explains how to use the driver, and provides a detailed guide to the programmatic interface for executing each Control Operator Driver routine. For quick reference, a section at the end of the chapter summarizes the data structures, constants, and routine parameters.

---

---

## Using the MacAPPC Control Operator Driver

Control operator routines permit a control operator to define and control certain components of a LU 6.2 node. They control session limits, activate and deactivate sessions, and define and display LUs, modes, and TPs.

To make the programmer's task easier, MacAPPC also provides a control operator program—the Administration program, described in Chapter 12—that handles all of the control operator functions. By using the Administration program, you do not need to access the control operator driver in the programs that you write; however, if you choose to do so, the control operator routines are fully described in this chapter.

---

## Security

MacAPPC supports two levels of security: session level and conversation level.

Session level LU-LU verification is used to verify the identity of each LU to its session-partner LU during activation of a session. A session between two LUs cannot be activated unless each one's view of the other is the same. In essence, each node must see a mirror image of the other node, both in configuration and in security, in order for a session to be successfully bound.

Conversation level access security information is carried on allocation requests in order for the receiving LU to verify the identity of the user ID and to control access to its resources. The security information includes a user ID together with a password or the already-verified indication. The information may also include a profile. This level of security takes place first at the LU level and then at the TP level. At the LU level, there can be zero or more user IDs, each with a password. For each user ID, there can be zero or more profiles.

The **profile** is an optional security feature that provides an additional element of structure and security to the network configuration. Some examples of profiles are department, store, corporation, section, division, function, building, location, and code designation.

If the allocation request passes the LU-level security it must then pass the security requirement of the TP level. Security checking may be waived at the TP level or consist of checking any combination of the user ID and the profile.



At the local LU level, conversation-level security may be turned on and off by radio button selection when you are using the Configuration program. The security consists of lists of authorized user IDs, passwords, and profiles that are checked and confirmed before a session bind may occur. These lists are created and edited for a local LU. The term *already verified* indicates that the local LU will accept conversation-level access without password checking as a result of having previously (already) verified the password during an earlier bind.

At the TP level, the security consists of lists of IDs, passwords, and profiles that are selected from the local LU lists.

The resource-access authorization list that is verified during security checking consists of the set of user IDs, passwords, and profiles that are specified for a specific LU or TP.

---

---

## MacAPPC control operator routines

MacAPPC control operator routines define, display, and control the following LU components:

- ☐ local LU
- ☐ remote LU
- ☐ mode
- ☐ transaction program (TP)

The routines are divided into the following categories:

- ☐ change-number-of-session (CNOS) routines, which control session limits for modes
- ☐ session control routines, which activate and deactivate sessions over modes
- ☐ LU definition routines, which define, display, and delete LU components

---

---

## Control operator CNOS routines

This section describes the MacAPPC control operator CNOS routines. These routines manipulate session limits for modes.



---

## COChangeSessionLimit

### Summary

The `COChangeSessionLimit` routine changes the session limits and polarities for parallel-session connections between the local LU and the remote LU over the specified mode. It cannot be used, however, when session limits are 0. To raise limits from 0, use the `COInitializeSessionLimit` routine. As a result of changes made using this routine, additional LU-LU sessions with the specified mode name can be activated or deactivated.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002A	long	→	coCVCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName
00AE	word	→	coCurMaxSess
00B4	word	→	coCurMinFirstSpkrs
00BA	word	→	coCurMinBdrs
0094	byte	→	coRespType

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coCVCBPtr** (supplied) specifies a pointer to a Conversation Control Block (CVCB). The block must be as long as the value specified by the `kCVCBSize` constant. You must supply a new CVCB each time your application executes the `COChangeSessionLimit` routine.

**coRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU. The string length must not be greater than the value of the `kMaxName` constant.

**coModeName** (supplied) specifies a pointer to a string that contains the name of the mode to which the change applies. The string length must not be greater than the value of the `kMaxName` constant. `COChangeSessionLimit` cannot be used to change the reserved `SNASVCMG` mode.

**coCurMaxSess** (supplied) specifies the requested limit for the maximum number of sessions allowed between the local LU and the remote LU over the specified mode. It must be greater than 0 and cannot exceed the configured maximum. Also, it cannot be less than the sum of the values of the `coCurMinFirstSpkrs` and `coCurMinFirstBdrs` parameters. The remote LU can negotiate the value to a lower value (which still must be greater than 0).

**coCurMinFirstSpkrs** (supplied) specifies the minimum number of sessions that can be activated as first-speaker sessions (contention winner) for the local LU. The remote LU can negotiate this value to some lower number.



**coCurMinBdrs** (supplied) specifies the minimum number of sessions that can be activated as bidder sessions (contention loser) for the local LU. The remote LU can negotiate this value to some lower number.

**coRespType** (supplied) specifies which LU (local or remote) is responsible for deactivating any active sessions if the new session limit decreases the number available. The sessions are deactivated when they are not allocated to conversations. The following values are defined:

kSrcResp indicates that the local LU (source) is responsible.

kTgtResp indicates that the remote LU (target) is responsible.

If the number of allowed sessions is not being decreased, this parameter is ignored.

#### Notes

This routine does not terminate active conversations. If a session is to be deactivated and is allocated to an active conversation, the responsible LU waits until the conversation is deallocated, and then deactivates the session.

#### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete



---

## COInitializeSessionLimit

### Summary

The `COInitializeSessionLimit` routine establishes the initial session limits and the polarities for single-session or parallel-session connections between the local LU and the remote LU over the specified mode. It can only be executed when the session limits are 0. As a result of initialization made using this routine, one or more LU-LU sessions with the specified mode name can be activated.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002A	long	→	coCVCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName
00AE	word	→	coCurMaxSess
00B4	word	→	coCurMinFirstSpkrs
00BA	word	→	coCurMinBdrs

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coCVCBPtr** (supplied) specifies a pointer to a Conversation Control Block (CVCB) whose length is equal to the value of the `kCVCBSize` constant. You must supply a new CVCB each time your application executes the `COInitializeSessionLimit` routine.

**coRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU to which the session limits apply. The string length must not be greater than the value of the `kMaxName` constant.

**coModeName** (supplied) specifies a pointer to a string that contains the name of the mode to be initialized. The string length must not be greater than the value of the `kMaxName` constant. Use `COInitializeSessionLimit` to initialize the reserved `SNASVCMG` mode.

**coCurMaxSess** (supplied) specifies the requested limit for the maximum number of sessions allowed between the local LU and the remote LU over the specified mode. It must be greater than 0 and cannot exceed the configured maximum. Also, it cannot be less than the sum of the values of the `coCurMinFirstSpkrs` and `coCurMinFirstBdrs` parameters. The remote LU can negotiate the value to a lower value (which still must be greater than 0).

❖ *Note:* For the `SNASVCMG` mode, the session limit must be 2.



**coCurMinFirstSpkrs** (supplied) specifies the minimum number of sessions that can be activated as first-speaker sessions (contention winner) for the local LU. The remote LU can negotiate this value to some lower number.

❖ *Note:* For the SNASVCMG mode, the first-speaker session minimum must be 1.

**coCurMinBdrs** (supplied) specifies the minimum number of sessions that can be activated as bidder sessions (contention loser) for the local LU. The remote LU can negotiate this value to some lower number.

❖ *Note:* For the SNASVCMG mode, the bidder session minimum must be 1.

<b>Result code</b>	
appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete



---

## COProcessSessionLimit

### Summary

The `COProcessSessionLimit` routine can only be executed by the CNOS transaction program to initiate a remote LU response to a request to change the number of sessions.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002A	long	→	coCVCBPtr
0032	long	⇒	coRmtLUName
0036	long	⇒	coModeName

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coCVCBPtr** (supplied) specifies a pointer to a Conversation Control Block (CVCB) whose length is equal to the value of the `kCVCBSize` constant. You must supply a new CVCB each time your application executes the `COProcessSessionLimit` routine.

**coRmtLUName** (supplied/modified) specifies a pointer to space where the name of the remote LU can be returned. The space must be at least as large as the value of the `kMaxName` constant plus 1 byte.

**coModeName** (supplied/modified) specifies a pointer to space where the name of the mode can be returned. The space must be at least as large as the value of the `kMaxName` constant plus 1 byte. A NULL string indicates that all modes are being reset.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete



---

## COResetSessionLimit

### Summary

The `COResetSessionLimit` routine resets the session limits for a given mode, or for all modes between the local LU and remote LU. All active sessions with the specified mode name or all modes are deactivated once conversation activity is over.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002A	long	→	coCVCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName
0094	byte	→	coRespType
0072	byte	→	coDrainSrc
0073	byte	→	coDrainTgt
0074	byte	→	coForceRst

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coCVCBPtr** (supplied) specifies a pointer to a Conversation Control Block (CVCB) whose length is equal to the value of the `kCVCBSize` constant. You must supply a new CVCB each time your application executes the `COResetSessionLimit` routine.

**coRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU. The string length must not be greater than the value of the `kMaxName` constant.

**coModeName** (supplied) specifies a pointer to a string that contains the name of the single mode to be reset. The string length must not be greater than the value of the `kMaxName` constant. Use the `COResetSessionLimit` routine to reset the reserved `SNASVCMG` mode. To specify all modes, set `coModeName` to `NIL`.

**coRespType** (supplied) specifies which LU (local or remote) is responsible for deactivating any active sessions if the new session limit decreases the number available. The sessions are deactivated when they are not allocated to conversations. The following values are defined:

`kSrcResp` indicates that the local LU (source) is responsible.

`kTgtResp` indicates that the remote LU (target) is responsible.

If the number of allowed sessions is not being decreased, this parameter is ignored. For a single-session connection, or for `SNASVCMG`, this parameter does not apply.

**coDrainSrc** (supplied) specifies whether or not the local LU (source) is allowed to honor any pending allocation requests before deactivating sessions. If the value is `FALSE`, it must deactivate sessions as soon as the current conversations are deallocated. If it has a value of `TRUE`, it can continue to allocate the sessions to conversations as long as allocation requests are outstanding.



**coDrainTgt** (supplied) specifies whether or not the remote LU (target) is allowed to honor any pending allocation requests before deactivating sessions. If the value is FALSE, it must deactivate sessions as soon as the current conversations are deallocated. If the value is TRUE, it can continue to allocate the sessions to conversations as long as allocation requests are outstanding.

**coForceRst** (supplied) specifies whether or not the local LU (source) is allowed to force the resetting of the session limits when certain error conditions occur. If the value is FALSE, the session limit is to be reset only upon successful completion of CNOS. If the value is TRUE, the session limit is to be reset when CNOS negotiations are successfully completed or when CNOS negotiations are unsuccessful due to certain error conditions. This allows the mode to be reset even if the line between the two LUs has been broken. For a single-session connection, or for SNASVCMG, this parameter does not apply.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete



---

---

## **Control operator session control routines**

This section describes the MacAPPC control operator session control routines. These routines activate and deactivate sessions.



---

## COActivateSession

### Summary

The COActivateSession routine activates a session to the remote LU using the properties defined by the specified mode. If the maximum number of first-speaker sessions has not been reached, the session is activated as a first-speaker session (contention winner). Otherwise, the session is activated as a bidder session (contention loser).

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU. The string length must not be greater than the value of the kMaxName constant.

**coModeName** (supplied) specifies a pointer to a string that contains the name of the mode defining the desired properties of the session. The string length must not be greater than the value of the kMaxName constant. COActivateSession can be used to activate a session over single-session or parallel-session connections and over the reserved SNASVCMG mode.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

CODEactivateSession



---

## CODeactivateSession

### Summary

The CODeactivateSession routine deactivates the specified session, either immediately or when the current conversation that was allocated to the session is deallocated.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName
006A	long	→	coSessID
0092	byte	→	coDeactType

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU. The string length must not be greater than the value of the `kMaxName` constant.

**coModeName** (supplied) specifies a pointer to a string that contains the mode name under which the session is active. The string length must not be greater than the value of the `kMaxName` constant.

**coSessID** (supplied) specifies the identifier of the session to be deactivated.

**coDeactType** (supplied) specifies the type of deactivation. The following values are defined:

`kNormalDeact` specifies that the session is to be deactivated when the current conversation that was allocated to the session is deallocated (if there is no conversation allocated to the session, it is deactivated immediately).

`kCleanupDeact` specifies that the session is to be deactivated immediately, regardless of any conversation that may be allocated to the session.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

COActivateSession



---

---

## **Control operator LU definition routines**

This section describes the MacAPPC control operator LU definition routines. These routines define, display, and delete LU components.



---

## CODefineLocalLU

### Summary

The CODefineLocalLU routine defines a local LU and its various attributes. To use it for this purpose, the routine should be executed prior to any network activity on the part of the local LU. CODefineLocalLU defines five areas:

- ☐ the local LU name
- ☐ the LU ID
- ☐ a network name and qualifier for the local LU
- ☐ a session limit for the LU
- ☐ conversation-level security information

This routine is used to initialize and modify operating parameters that are located in the new or existing local LU. The first time it is executed, this routine initializes the local LU definition with default values and updates it with the specified operating parameters. On subsequent executions this routine updates the local LU definition with the data supplied for the specified operating parameters. If a parameter is not specified, the value currently defined for that parameter remains unchanged. This routine can be executed by any transaction program that has the privilege to execute definition routines.

If the session limits or the session count for any modes defined for this local LU are greater than 0, the network name, qualifier, and LU ID cannot be updated.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
007E	byte	→	coNetNameOp
003E	long	→	coNetName
007F	byte	→	coNetQualOp
0042	long	→	coNetQual
0081	byte	→	coSecOp
0052	long	→	coUserName
0056	long	→	coUserPswd
005A	long	→	coUserProf
0070	word	→	coWaitTime
00A8	word	→	coDefLUMaxSess
009E	word	→	coMaxTP
006E	byte	→	coLclLUID
008F	byte	→	coConvSecType

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).



**coLclLUName** (supplied) specifies a pointer to a string that contains the name of the local LU being defined or modified. The string length must not be greater than the value of the **kMaxName** constant, and the characters in the string must be of symbol-string type A.

**coNetNameOp** (supplied) specifies the operation to be performed using the **coNetName** parameter. The following values are defined:

**kIgnoreParam** indicates that the network name is not specified.

**kReplaceParam** indicates that the network name replaces the currently defined network name.

**kDeleteParam** indicates that the network name is to be deleted.

**coNetName** (supplied) specifies a pointer to a string that contains the network name of the local LU. The string length must not be greater than the value of the **kMaxName** constant. This is the name by which the local LU is known throughout the network. If not defined, the local LU name is used as the network name. This parameter is required if the **coNetNameOp** parameter specifies the value of the **kReplaceParam** or **kDeleteParam** constant. The characters contained in this string must be of symbol-string type A with the exception of an asterisk (\*), which indicates that the network name for this remote LU is blank. If the network name is blank, any name is accepted from the remote LU in a session-activation request. A name of eight space characters is sent to the remote LU on session-activation requests.

**coNetQualOp** (supplied) specifies the operation to be performed using the **coNetQual** parameter. The following values are defined:

**kIgnoreParam** indicates that the network qualifier is not specified.

**kReplaceParam** indicates that the network qualifier replaces the currently defined network qualifier.

**kDeleteParam** indicates that the network qualifier is to be deleted.

**coNetQual** (supplied) specifies a pointer to a string that contains the network qualifier of the network name. The string length must not be greater than the value of the **kMaxName** constant. This is an optional parameter that can be defined for cross-network communication to give uniqueness to the network name. This parameter is required if the **coNetQualOp** parameter is set to the **kReplaceParam** or **kDeleteParam** constant.

**coSecOp** (supplied) specifies the operation to be performed using a combination of the **coUserName**, **coUserPswd**, and **coUserProf** parameters. The following values are defined:

**kIgnoreParam** indicates that the security parameters are not specified.

**kAddParam** indicates that the security parameters are to be added.

**kDeleteParam** indicates that the security parameters are to be deleted.

**coUserName** (supplied) specifies a pointer to a string that contains a user ID. The string length must not be greater than the value of the **kMaxSecName** constant. If any security parameters are to be added or deleted, a user ID must be given. If security parameters are to be added, see the **coUserPswd** and **coUserProf** parameters for additional restrictions. If the user ID and associated profiles are to be deleted, provide the user ID with no additional security parameters; that is, you should specify **coUserPswd** and **coUserProf** as NIL pointers.



**coUserPswd** (supplied) specifies a pointer to a string that contains a password for the user ID specified in the **coUserName** parameter. The string length must not be greater than the value of the **kMaxSecName** constant. If security parameters are to be added, a password must be provided when the user ID has not yet been defined. If the security parameters are to be deleted, no password should be given. If a new password is given, it replaces the old password.

**coUserProf** (supplied) specifies a pointer to a string that specifies a profile for the preceding user ID. The string length must not be greater than the value of the **kMaxSecName** constant. If the security parameters are to be added, a profile is optional. If a profile is given, it is added to the list of profiles for that user ID. If the security parameters are to be deleted, specify the profile to be deleted; if no single profile is specified, all profiles are deleted for that user ID.

**coWaitTime** (supplied) specifies in terms of seconds a value for the amount of time the LU waits for routine completion. The valid range is 1 to 3600. The default value for this parameter is 60. The **kIgnoreParam** constant indicates that the parameter is not specified.

**coDefLUMaxSess** (supplied) specifies the value of the maximum number of sessions that can be active at the local LU at one time. The valid session limit range is 1 to 254. The value specified must be greater than or equal to the largest session limit for any modes defined for this local LU. Specifying 0 indicates that this value should be ignored. The default value for this parameter is 0. The **kIgnoreParam** constant indicates that the parameter is not specified.

❖ *Note:* Session limits are currently maintained only at a mode level. This value is stored, but not used.

**coMaxTP** (supplied) specifies a value for the maximum number of transaction programs that can be attached at one time. The valid range is between 1 and 255. The default value for this parameter is 5. The **kIgnoreParam** constant indicates that the parameter is not specified.

**coLclLUID** (supplied) specifies a value from 1 to 254 that matches the destination address field (DAF) that is received on the Activate Logical Unit (ACTLU) request sent from a host. Within a node, each local LU must have this parameter set to a unique value. This parameter must be specified the first time the routine is executed and the local LU is initialized. On subsequent executions, **kIgnoreParam** indicates that the parameter is not specified.

**coConvSecType** (supplied) specifies whether or not conversation-level security is enabled for the local LU. If users are defined to the local LU, conversational-level security is performed on incoming allocation requests. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** specifies that LU security is not supported. This value is the default for this parameter.

**kFuncSupp** specifies that LU security is supported.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **CODisplayLocalLU**



---

## CODefineMode

### Summary

The CODefineMode routine defines a group of session characteristics between a local LU and a remote LU. These characteristics include the following:

- ☐ the send and receive pacing count
- ☐ the lower and upper bound for the maximum request/response unit (RU)
- ☐ the synchronization level that can be used over this mode
- ☐ single-session reinitiation responsibility
- ☐ the maximum number of sessions that can be active at one time
- ☐ the minimum number of first-speaker and prebound first-speaker sessions that can be initialized for this mode

This routine is used to initialize and modify operating parameters that are located in the new or existing mode. The first time it is executed, this routine initializes the mode definition with default values and updates it with the specified operating parameters. On subsequent executions this routine updates the mode definition with the data supplied for the specified operating parameters. If a parameter is not specified, the value currently defined for that parameter remains unchanged. This routine can be executed by any transaction program that has the privilege to execute definition routines.

If the session limits or the session count for the mode is greater than 0, only the coDefPBFirstSpkrs parameter can be updated.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
0036	long	→	coModeName
004A	long	→	coALSName
00A0	word	→	coSendPacing
00A2	word	→	coRcvPacing
00AC	word	→	coDefMaxSess
00B2	word	→	coDefMinFirstSpkrs
00BE	word	→	coDefPBFirstSpkrs
00A4	word	→	coMaxRUHiBound
00A6	word	→	coMaxRULoBound
0090	byte	→	coSyncType
009A	byte	→	coReinitType
008C	byte	→	coSessCrypt
0084	byte	→	coQueueBINDs
008D	byte	→	coBlankMode



## Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coLclLUName** (supplied) specifies a pointer to a string that contains the local LU name for which a mode is being defined or modified. The string length must not be greater than the value of the **kMaxName** constant, and the characters in the string must be of symbol-string type A. This name must correspond to that of a previously defined LU.

**coRmtLUName** (supplied) specifies a pointer to a string that contains the remote LU name for which a mode is being defined or modified. The string length must not be greater than the value of the **kMaxName** constant, and the characters in the string must be of symbol-string type A. This name must correspond to that of a previously defined remote LU.

**coModeName** (supplied) specifies a pointer to a string that contains the name of the mode being defined or modified. The string length must not be greater than the value of the **kMaxName** constant, and the characters in the string must be of symbol-string type A. This name is used to refer to a specific mode when updating its parameters. **SNASVCMG** is a reserved name and must not be used.

**coALSName** (supplied) specifies a pointer to a string that contains either the name of a previously defined adjacent link station or the character string **LOCAL**. The string length must not be greater than the value of the **kMaxName** constant. Specify **LOCAL** to indicate that the local LU and the remote LU are found in the same node. The first time this routine is executed, it is a required parameter. On subsequent executions, a **NIL** pointer indicates that the currently defined value of the ALS Name remains unchanged.

**coSendPacing** (supplied) specifies the number of requests the local LU can send before receiving a pacing response. This is used to determine the send window size during session activation. The valid range for this field is 0 to 21. The default value for this parameter is 7. The **kIgnoreParam** constant indicates that the parameter is not specified.

**coRcvPacing** (supplied) specifies the number of requests the local LU can receive before sending a pacing response. This is used to determine the receive window size during session activation. The valid range for this field is 0 to 21. The default value for this parameter is 7. The **kIgnoreParam** constant indicates that the parameter is not specified.

**coDefMaxSess** (supplied) specifies the maximum number of sessions that can be active at a given time for this mode. If a value greater than 1 is specified, parallel sessions must be supported by the remote LU. The valid range is 1 to 254. The default value for this parameter is 6. The **kIgnoreParam** constant indicates that the parameter is not specified.

**coDefMinFirstSpkrs** (supplied) specifies the number of first-speaker sessions for this mode. The valid range is 1 to the value specified for the **coLUMaxSess** parameter. The default value for this parameter is 3. The **kIgnoreParam** constant indicates that the parameter is not specified.



**coDefPBFirstSpkrs** (supplied) specifies the minimum number of first-speaker sessions that are automatically activated when session limits are initialized. The valid range is 1 to the value specified for the **coDefMinFirstSpkrs** parameter. The default value for this parameter is 3. The **kIgnoreParam** constant indicates that the parameter is not specified.

**coMaxRUHiBound** (supplied) specifies the upper bound of the maximum RU size that can be sent or received across this mode. This field is used in conjunction with the **coMaxRULowerBound** parameter to determine the maximum RU size possible during session activation. The valid range is 256 to 4096. The default value for this parameter is 1024. The **kIgnoreParam** constant indicates that the parameter is not specified.

**coMaxRULoBound** (supplied) specifies the lower bound of the maximum RU size that can be sent or received across this mode. This field is used in conjunction with the **coMaxRUHiBound** parameter to determine the maximum RU size possible during session activation. The valid range is 8 to 256. The default value for this parameter is 256. The **kIgnoreParam** constant indicates that the parameter is not specified.

**coSyncType** (supplied) specifies the synchronization levels that can be used by conversations using sessions over this mode. The following values are defined:

**kIgnoreParam** specifies that the parameter is not specified.

**kConfirmModeSync** specifies that transaction programs can perform either no confirmation processing or perform confirmation processing, but cannot perform sync-point processing on conversations using this mode. This value is the default for this parameter.

**kSyncPtModeSync** specifies that transaction programs can perform no confirmation processing, can perform confirmation processing, or can perform sync-point processing on conversations using this mode.

❖ *Note:* At the time of publication, sync-point services were not supported.

**coReinitType** (supplied) specifies responsibility for single-session reinitiation. It can only be specified when the remote LU does not support parallel sessions. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kOperInit** indicates that an operator from either the local or remote LU attempts session reinitiation. Neither LU attempts automatic reinitiation.

**kPriLUInit** indicates that the primary LU automatically attempts the reinitiation.

**kSecLUInit** indicates that the secondary LU automatically attempts the reinitiation.

**kEitherLUInit** indicates that either the primary or the secondary LU automatically attempts the reinitiation. This value is the default for this parameter.



**coSessCrypt** (supplied) specifies whether or not session-level cryptography is supported for this mode. The following values are defined:

kIgnoreParam indicates that parameter is not specified.

kFuncNotSupp indicates that session-level cryptography is not supported. This value is the default for this parameter.

kFuncNotSupp indicates that the session-level cryptography is supported.

❖ *Note:* At the time of publication, session-level cryptography was not supported.

**coQueueBINDs** (supplied) specifies whether or not BINDs sent across this mode can be queued if the partner is not able to accept them. The following values are defined:

kIgnoreParam indicates that the parameter is not specified.

kFuncNotSupp indicates that BINDs cannot be queued.

kFuncSupp indicates that BINDs can be queued. This value is the default for this parameter.

**coBlankMode** (supplied) specifies whether a null mode name is sent across the link. The following values are defined:

kIgnoreParam indicates that the parameter is not specified.

kFuncNotSupp indicates that the null mode names are not supported. The actual mode name is sent across the link. This value is the default for this parameter.

kFuncSupp indicates that the null mode names are supported. The mode name sent across the link is eight space characters. The kFuncSupp constant can be provided for only one mode per remote LU.

<b>Result code</b>	appcNoErr	Routine succeeded
	appcFail	Routine failed; look in appcHiResult and appcLoResult
	appcExec	Routine executing; asynchronous request not complete

**See also** CODisplayMode, CODEfineLocalLU, CODEfineRemoteLU



---

## CODefineRemoteLU

### Summary

The CODefineRemoteLU routine defines the parameters for a remote LU that is configured as a partner for the specified local LU. These parameters include the following values:

- ☐ the name by which the remote LU is locally known
- ☐ the network name
- ☐ an LU password used for session-level verification
- ☐ an indication of whether or not session-initiation requests can be queued
- ☐ an indication of whether or not parallel sessions are supported
- ☐ an indication of whether or not access-security information parameters are accepted on an allocation request

This routine is used to initialize and modify operating parameters that are located in the new or existing remote LU. The first time it is executed, this routine initializes the remote LU definition with default values and updates it with the specified operating parameters. On subsequent executions this routine updates the remote LU definition with the data supplied for the specified operating parameters. If a parameter is not specified, the value currently defined for that parameter remains unchanged. This routine can be executed by any transaction program that has the privilege to execute definition routines.

If the session limits or the session count for any modes defined for this remote LU are greater than 0, no parameters can be updated.

Parameters	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0026	long	→	coTPCBPtr
	002E	long	→	coLclLUName
	0032	long	→	coRmtLUName
	007E	byte	→	coNetNameOp
	003E	long	→	coNetName
	007F	byte	→	coNetQualOp
	0042	long	→	coNetQual
	0046	long	→	coCPName
	004E	long	→	coCNOSALSName
	0080	byte	→	coLUPswdOp
	005E	long	→	coLUPswd
	008E	byte	→	coQueueINITs
	0083	byte	→	coParSess
	0097	byte	→	coLclSecAcc



## Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coLclLUName** (supplied) specifies a pointer to a string that contains the local LU name for which a remote LU is being defined. The string length must not be greater than the value of the `kMaxName` constant, and the characters in the string must be of symbol-string type A.

**coRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU being defined or modified. The string length must not be greater than the value of the `kMaxName` constant, and the characters in the string must be of symbol-string type A.

**coNetNameOp** (supplied) specifies the operation to be performed using the `coNetName` parameter. The following values are defined:

`kIgnoreParam` indicates that the network name is not specified.

`kReplaceParam` indicates that the network name replaces the currently defined network name.

`kDeleteParam` indicates that the network name is to be deleted.

**coNetName** (supplied) specifies a pointer to a string that contains the network name by which the remote LU is known throughout the network. The string length must not be greater than the value of the `kMaxName` constant. If this parameter is not defined, the remote LU name is used as the network name. The characters contained in this string must be of symbol-string type A with the exception of an asterisk (\*), which indicates that the network name for this remote LU is blank. This parameter is required if the `coNetNameOp` parameter specifies the `kReplaceParam` or `kDeleteParam` constant.

**coNetQualOp** (supplied) specifies the operation to be performed using the `coNetQual` parameter. The following values are defined:

`kIgnoreParam` indicates that the network qualifier is not specified.

`kReplaceParam` indicates that the network qualifier replaces the currently defined network qualifier.

`kDeleteParam` indicates that the network qualifier is to be deleted.

**coNetQual** (supplied) specifies a pointer to a string that contains the network qualifier of the network name. The string length must not be greater than the value of the `kMaxName` constant, and the characters in the string must be of symbol-string type A. This is an optional field that can be defined for cross-network communications to give uniqueness to the network name. This parameter is required if the `coNetQualOp` parameter specifies the value of the `kReplaceParam` or `kDeleteParam` constant.

**coCPName** (supplied) specifies a pointer to a string that contains the name of a previously defined control point where the remote LU is located. The string length must not be greater than the value of the `kMaxName` constant. This parameter is needed only if the control point is a host.

**coCNOSALSName** (supplied) specifies a pointer to a string that contains either the name of a previously defined adjacent link station or the character string LOCAL. The string length must not be greater than the value of the `kMaxName` constant. This is the name of the adjacent link station that handles CNOS negotiations. Specify LOCAL to indicate that the local LU and the remote LU are found in the same node. If parallel sessions are supported, this is a required parameter.



**coLUPswdOp** (supplied) specifies the operation to be performed using the **coLUPswd** parameter. The following values are defined:

**kIgnoreParam** indicates that the LU-LU password is not specified.

**kReplaceParam** indicates that the LU-LU password replaces the currently defined LU-LU password.

**kDeleteParam** indicates that the LU-LU password is to be deleted.

**coLUPswd** (supplied) specifies a pointer to a string that contains the LU-LU password defined for the local LU by the remote LU. The string must be exactly as long as the value of the **kMaxLUPswd** constant, and must consist of hexadecimal characters. This parameter is used to provide session-level verification during session activation. This parameter is required if the **coLUPswdOp** parameter is set to the **kReplaceParam** or **kDeleteParam** constant.

**coQueueINITS** (supplied) indicates whether or not the system services control point (SSCP) should queue session-initiation requests if the remote LU is not able to accept them. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that the SSCP does not queue session-initiation requests.

**kFuncSupp** indicates that the SSCP queues session-initiation requests. This value is the default for this parameter.

**coParSess** (supplied) specifies whether or not the remote LU supports parallel sessions. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that parallel sessions are not supported.

**kFuncSupp** indicates that parallel sessions are supported. This value is the default for this parameter.

**coLclSecAcc** (supplied) specifies the access-security information the local LU accepts for the remote LU being defined. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kNoSecAcc** indicates that no access-security information is accepted. This value is the default for this parameter.

**kConvSecAcc** indicates that access-security information is accepted but the already-verified indication as set by the allocation request is not accepted.

**kVerifSecAcc** indicates that access-security information is accepted and the already-verified indication as set by the allocation request is accepted.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **CODisplayRemoteLU**, **CODefineLocalLU**



---

## CODefineTP

### Summary

The CODefineTP routine defines a transaction program and its characteristics for the specified local LU. The characteristics that the routine defines include the following:

- ☐ the status of the transaction program
- ☐ conversation type
- ☐ synchronization level
- ☐ security required
- ☐ data required for program initialization parameters (PIP)
- ☐ an indication of whether or not Function Management Header (FMH) data support is provided
- ☐ the category of control operator routines the transaction program is allowed to execute

This routine is used to initialize and modify operating parameters that are located in the new or existing TP. The first time it is executed, this routine initializes the TP definition with default values and updates it with the specified operating parameters. On subsequent executions this routine updates the TP definition with the data supplied for the specified operating parameters. If a parameter is not specified, the value currently defined for that parameter remains unchanged. This routine can be executed by any transaction program that has the privilege to execute definition routines.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
003A	long	→	coLclProgName
007E	byte	→	coNetNameOp
003E	long	→	coNetName
0052	long	→	coUserName
005A	long	→	coUserProf
008A	word	→	coPIPCount
0088	byte	→	coPIPCheck
0087	byte	→	coPIPReq
0086	byte	→	coFMHDataSupp
0082	byte	→	coLUWSupp
0096	byte	→	coPrivType
0099	byte	→	coSecReq
0081	byte	→	coSecOp
0093	byte	→	coEnableType
0091	byte	→	coConvType
0090	byte	→	coSyncType
0085	byte	→	coDataMapping



## Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coLclLUName** (supplied) specifies a pointer to a string that contains the local LU name for which a transaction program is being defined or modified. The string length must not be greater than the value of the **kMaxName** constant, and the characters in the string must be of symbol-string type A. This name must correspond to that of a previously defined LU.

**coLclProgName** (supplied) specifies a pointer to a string that contains the name of the transaction program being defined or modified. The string length must not be greater than the value of the **kMaxTPName** constant. Specify an asterisk (\*) to indicate that any transaction program can attach.

**coNetNameOp** (supplied) specifies the operation to be performed using the **coNetName** parameter. The following values are defined:

**kIgnoreParam** indicates that the network name is not specified.

**kReplaceParam** indicates that the network name replaces the currently defined network name.

**kDeleteParam** indicates that the currently defined network name is to be deleted.

**coNetName** (supplied) specifies a pointer to a string that contains the network name of the transaction program. The string length must not be greater than the value of the **kMaxTPName** constant, and the characters in the string must be hexadecimal. This field is required for service transaction programs, such as CNOS and DIA. When a request is received that specifies a program by a network name, the transaction program specified in the **coLclProgName** parameter is invoked.

**coUserName** (supplied) specifies a pointer to a string that contains a user ID. The string length must not be greater than the value of the **kMaxSecName** constant. A user ID must be indicated if the security requires user IDs on the resource-access authorization list.

**coUserProf** (supplied) specifies a pointer to a string that contains a profile. The string length must not be greater than the value of the **kMaxSecName** constant. A profile must be indicated if the security requires profiles on the resource-access authorization list.

**coPIPCount** (supplied) specifies the number of program initialization parameter (PIP) subfields that may be required on allocation requests that start this transaction program. The valid range for this parameter is a number between 0 and 256. The **kIgnoreParam** constant indicates that the parameter is not specified. The default value for this parameter is 0.

❖ *Note:* Support for parameter number checking is not implemented. This value is stored, but not used.



**coPIPCheck** (supplied) specifies whether or not an allocation request is rejected if the value specified in the **coPIPCount** parameter does not match the number of PIP subfields supplied on the request. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that the request is not rejected. This value is the default for this parameter.

**kFuncSupp** indicates that the request is rejected.

❖ *Note:* Support for parameter number checking is not implemented. This value is stored, but not used.

**coPIPReq** (supplied) specifies whether or not PIP data is required on allocation requests designating this transaction program. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that PIP data is not allowed on allocation requests.

**kFuncSupp** indicates that PIP data is allowed on allocation requests. This value is the default for this parameter.

**coFMHDataSupp** (supplied) specifies whether or not Function Management Header (FMH) data can be received and sent by this transaction program. This parameter applies only when the **coConvType** parameter is set to the **kMappedTPConv** or **kEitherTPConv** constant. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that FMH data cannot be transmitted or received by this transaction program. This value is the default for this parameter.

**kFuncSupp** indicates that FMH data can be transmitted or received by this transaction program.

**coLUWSupp** (supplied) specifies whether or not the transaction program assigns logical-unit-of-work IDs for each transaction executed by this program. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that logical-unit-of-work IDs will not be assigned. This value is the default for this parameter.

**kFuncSupp** indicates that logical-unit-of-work IDs will be assigned.

**coPrivType** (supplied) specifies the type of control-operator routines that this transaction program can execute. Whether the program can execute basic or mapped routines is determined by the **coConvType** parameter. The following values are defined (any values except **kIgnoreParam** or **kNoPriv** can be combined in a logical OR operation):

**kIgnoreParam** indicates that the parameter is not specified.

**kNoPriv** indicates that the program is not allowed to execute routines that require a privilege to do so. This value is the default for this parameter.

**kCNOSPriv** indicates that the program is allowed to execute CNOS routines.

**kSessCtlPriv** indicates that the program is allowed to execute session-control routines.



**kDefinePriv** indicates that the program is allowed to execute definition routines.

**kDisplayPriv** indicates that the program is allowed to execute display routines.

**kSvcTPPriv** indicates that the program is allowed to execute an **MCAAllocate** or **BCAllocate** routine with the transaction program name designating a service transaction program, such as SNADS or DIA.

**coSecReq** (supplied) specifies the security verification allowed on an allocation request to start the program. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kNoSecReq** indicates that no security is required. If an allocation request does contain a user ID and password, conversation-level security is performed. This value is the default for this parameter.

**kConvSecReq** indicates that conversation-level security is required. Allocation requests designating this transaction program must carry a user ID and password. If the remote LU definition has the **coLclSecAcc** parameter defined as **kVerifSecAcc**, the allocation request can contain the already-verified indication in place of a password.

**kProfSecReq** indicates that resource-access security is required. The profile carried on the allocation request designating this transaction program must match a profile on the resource-access authorization list.

**kUserNameSecReq** indicates that resource-access security is required. The user ID carried on the allocation request designating this transaction program must match a user ID defined on the resource-access authorization list.

**kBothSecReq** indicates that resource-access security is required. The user ID and profile carried on the allocation request designating this transaction program must match a user ID and profile on the resource-access authorization list.

**coSecOp** (supplied) specifies the operation to be performed on the resource-access authorization list. For more information on resource-access authorization lists, see "Security" earlier in this chapter. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kAddParam** indicates that the security parameters are to be added.

**kDeleteParam** indicates that the security parameters are to be deleted.

**coEnableType** (supplied) specifies the response the LU makes when an allocation request is received that designates this transaction program. By changing the setting of this parameter, a program can be temporarily or permanently removed from network availability. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kEnableTP** indicates that the transaction program is available. This value is the default for this parameter.

**kDisableTempTP** indicates that the transaction program is temporarily unavailable. The response returned by the LU indicates retry is possible.

**kDisablePermTP** indicates that the transaction is permanently unavailable. The response returned by the LU indicates retry is not possible.



**coConvType** (supplied) specifies the conversation type allowed on an allocation request to start the program. The following values are defined:

kIgnoreParam indicates that the parameter is not specified.

kBasicTPConv indicates that the allocation request must specify a basic conversation.

kMappedTPConv indicates that the allocation request must specify a mapped conversation.

kEitherTPConv indicates that the allocation request can specify either a basic or mapped conversation. This value is the default for this parameter.

**coSyncType** (supplied) specifies the synchronization allowed on an allocation request to start the program. The following values are defined (any values except kIgnoreParam can be combined in a logical OR operation):

kIgnoreParam indicates that the parameter is not specified.

kNoTPSync indicates that the allocation request can specify no confirmation processing.

kConfirmTPSync indicates that the allocation request can specify confirmation processing. This value is the default for this parameter.

kSyncPtTPSync indicates that the allocation request can specify sync-point processing.

❖ *Note:* At the time of publication, sync-point services were not supported.

**coDataMapping** (supplied) specifies whether the transaction program is provided with data mapping support. The following values are defined:

kIgnoreParam indicates that the parameter is not specified.

kFuncNotSupp indicates that data mapping is not supported. This value is the default for this parameter.

kFuncSupp indicates that data mapping is supported.

<b>Result code</b>	appcNoErr	Routine succeeded
	appcFail	Routine failed; look in appcHiResult and appcLoResult
	appcExec	Routine executing; asynchronous request not complete

**See also** CODisplayTP, CODEfineLocalLU



---

## CODElete

### Summary

The CODElete routine deletes LU components as defined by the definition routines.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
003A	long	→	coLclProgName
0036	long	→	coModeName

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coLclLUName** (supplied) specifies a pointer to a string that contains a local LU name. The string length must not be greater than the value of the `kMaxName` constant. Specifying this parameter with no other parameters deletes the LU definition and associated partner definitions, mode definitions, and transaction program definitions.

**coRmtLUName** (supplied) specifies a pointer to a string that contains a remote LU name. The string length must not be greater than the value of the `kMaxName` constant. When this parameter is specified, the `coLclLUName` parameter must also be specified. When this parameter is specified in conjunction with the `coModeName` parameter, the mode definition is deleted. When specified without the mode name, the partner definition and all mode definitions associated with the partner definition are deleted.

**coLclProgName** (supplied) specifies a pointer to a string that contains a transaction program name. The string length must not be greater than the value of the `kMaxTPName` constant. When this parameter is specified, the `coLclLUName` parameter must also be specified, and all parameters associated with this transaction program name are deleted, including all user IDs and profiles.

**coModeName** (supplied) specifies a pointer to a string that contains a mode name. The string length must not be greater than the value of the `kMaxName` constant. When specified, it must be indicated in conjunction with the `coLclLUName` and `coRmtLUName` parameters.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
appcExec	Routine executing; asynchronous request not complete



---

## CODisplayLocalLU

### Summary

The CODisplayLocalLU routine returns local LU configuration information. This function can return information for the first LU defined for the node, the next LU, or a specific LU. For information on the meaning of the returned values, see the documentation for CODEfineLocalLU.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
0078	byte	→	coNextLclLUName
002E	long	⇒	coLclLUName
003E	long	⇒	coNetName
0042	long	⇒	coNetQual
007C	byte	→	coNextUserName
0052	long	⇒	coUserName
0056	long	⇒	coUserPswd
005A	long	⇒	coUserProf
0070	word	←	coWaitTime
00A8	word	←	coDefLUMaxSess
00AA	word	←	coActLUSess
009E	word	←	coMaxTP
006E	byte	←	coLclLUID
008F	byte	←	coConvSecType
0077	byte	←	coLUActive

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coNextLclLUName** (supplied) specifies that the information for the next local LU should be returned. This parameter is used only if the **coLclLUName** parameter is specified. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kNextEntry** indicates that the information for the next LU should be returned.

**coLclLUName** (supplied/modified) specifies a pointer to space where the name of the local LU can be returned. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If the pointer points to a NULL string, information on the first LU, including its name, is returned. If the pointer points to a name and the **coNextLclLUName** parameter is set to **kIgnoreParam**, information for the specified LU is returned. If the **coNextLclLUName** parameter is set to **kNextEntry**, information for the next LU, including its name, is returned.

**coNetName** (supplied/modified) specifies a pointer to space where the network name can be returned. The space must be at least as long as the value of the **kMaxTPName** constant plus 1 byte. If **coNetName** is NIL, the name is not returned.



**coNetQual** (supplied/modified) specifies a pointer to space where the network qualifier can be returned. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If this pointer is NIL, the qualifier is not returned.

**coNextUserName** (supplied) specifies that information for the next user ID defined should be returned. This parameter is used only if the **coUserName** parameter is specified. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kNextEntry** indicates that the information for the next user ID should be returned.

**coUserName** (supplied/modified) specifies a pointer to space where a string that contains the name of a user ID can be returned. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If the pointer points to a NULL string, the ID and related information for the first user found are returned. If the pointer points to a user ID and the **coNextUserName** parameter is set to the **kNextEntry** constant, information for the next user is returned. If the **coNextUserName** parameter is set to **kIgnoreParam**, information for the specified user is returned. If this pointer is NIL, the user ID is not returned.

**coUserPswd** (supplied/modified) specifies a pointer to space where a password can be returned. The space must be at least as long as the value of the **kMaxSecName** constant plus 1 byte. If this pointer is NIL, the password is not returned.

**coUserProf** (supplied/modified) specifies a pointer to space where a user profile can be returned. The space must be at least as long as the value of the **kMaxSecName** constant plus 1 byte. If the pointer points to a NULL string, the first profile associated with the returned user ID is returned. If a profile is supplied, the next profile is returned. If this pointer is NIL, the profile is not returned.

**coWaitTime** (returned) indicates the amount of time, in seconds, that the LU waits for routine completion.

**coDefLUMaxSess** (returned) indicates the value of the maximum number of sessions that can be active at the local LU simultaneously.

**coActLUSess** (returned) indicates the value of the number of sessions that are currently active for this local LU.

**coMaxTP** (returned) indicates the maximum number of transaction programs that can be attached simultaneously.

**coLclLUID** (returned) indicates the unique ID value for this local LU.

**coConvSecType** (returned) indicates whether or not conversation-level security is enabled for the local LU. The following values can be returned:

**kFuncSupp** indicates that security is enabled.

**kFuncNotSupp** indicates that security is not enabled.

**coLUActive** (returned) indicates whether the LU has been locally activated.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **CODefineLocalLU**



---

## CODisplayMode

### Summary

The `CODisplayMode` routine returns information for a mode configured between the specified local LU and the specified remote LU. To display session IDs for a particular mode, execute the `CODisplaySession` routine. For more information on the meaning of the values returned here, see the documentation for `CODefineMode`.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
007A	byte	→	coNextModeName
0036	long	⇒	coModeName
004A	long	⇒	coALSName
00A0	word	←	coSendPacing
00A2	word	←	coRcvPacing
00A4	word	←	coMaxRUHiBound
00A6	word	←	coMaxRULoBound
00AC	word	←	coDefMaxSess
00B2	word	←	coDefMinFirstSpkrs
00BE	word	←	coDefPBFirstSpkrs
00B8	word	←	coDefMinBdrs
00AE	word	←	coCurMaxSess
00B4	word	←	coCurMinFirstSpkrs
00BA	word	←	coCurMinBdrs
00B0	word	←	coActSess
00B6	word	←	coActFirstSpkrs
00BC	word	←	coActBdrs
0090	byte	←	coSyncType
009A	byte	←	coReinitType
008C	byte	←	coSessCrypt
0084	byte	←	coQueueBINDs
008D	byte	←	coBlankMode
009C	byte	←	coTermCount
0075	byte	←	coDrainLclLU
0076	byte	←	coDrainRmtLU

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coLclLUName** (supplied) specifies a pointer to a string that contains the local LU name for which the mode is being displayed. The string length must not be greater than the value of the `kMaxName` constant.

**coRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU for which the mode is being displayed. The string length must not be greater than the value of the `kMaxName` constant.



**coNextModeName** (supplied) specifies that information for the next mode should be returned. This parameter is used only if **coModeName** is specified. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kNextEntry** indicates that the information for the next mode should be returned.

**coModeName** (supplied/modified) specifies a pointer to space where the name of the mode can be returned. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If the pointer points to a NULL string, information on the first mode for this local LU/remote LU is returned. This information includes the mode name. If a name is specified and the **coNextModeName** parameter is set to **kIgnoreParam**, information for the specified mode is returned. If **coNextModeName** is set to **kNextEntry**, information for the next mode, including its name, is returned.

**coALSName** (supplied/modified) specifies a pointer to space where the name of the adjacent-link station used by this mode can be returned. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If this pointer is NIL, the name is not returned.

**coSendPacing** (returned) indicates the number of requests that the local LU can send before receiving a pacing response.

**coRcvPacing** (returned) indicates the number of requests that the local LU can receive before sending a pacing response.

**coMaxRUHiBound** (returned) indicates the upper bound of the maximum RU size that can be sent or received across this mode.

**coMaxRULoBound** (returned) indicates the lower bound of the maximum RU size that can be sent or received across this mode.

**coDefMaxSess** (returned) indicates the defined session limit for this mode.

**coDefMinFirstSpkrs** (returned) indicates the defined minimum number of first-speaker sessions for this mode.

**coDefPBFirstSpkrs** (returned) indicates the defined number of first-speaker sessions that are automatically activated when sessions limits are initialized.

**coDefMinBdrs** (returned) indicates the defined minimum number of bidder sessions for this mode.

**coCurMaxSess** (returned) indicates the maximum number of sessions that can be active simultaneously for this mode. This is the current limit, set by way of a CNOS exchange with the remote LU, or set locally. If it is zero, the mode is unavailable for conversation use.

**coCurMinFirstSpkrs** (returned) indicates the minimum number of first-speaker sessions that can be active simultaneously for this mode. This is the current limit, set by way of a CNOS exchange with the remote LU, or set locally.

**coCurMinBdrs** (returned) indicates the current minimum number of bidder sessions for this mode. This is the current limit, set by way of a CNOS exchange with the remote LU, or set locally.

**coActSess** (returned) indicates the total number of sessions that are currently active for this mode.



**coActFirstSpkrs** (returned) indicates the number of first-speaker sessions that are currently active for this mode.

**coActBdrs** (returned) indicates the number of bidder sessions that are currently active for this mode.

**coSyncType** (returned) indicates the synchronization levels that conversations using sessions over this mode can use. The following values can be returned:

**kConfirmModeSync** indicates that transaction programs either cannot perform confirmation processing or can perform confirmation processing, but cannot perform sync-point processing on conversations using this mode.

**kSyncPtModeSync** indicates that transaction programs either cannot perform confirmation processing, can perform confirmation processing, or can perform sync-point processing on conversations using this mode.

❖ *Note:* At the time of publication, sync-point services were not supported.

**coReinitType** (returned) indicates the responsibility for reinitiation of a single session. The following values can be returned:

**kOperInit** indicates that an operator from either the local or remote LU attempts session reinitiation. Neither LU will attempt automatic reinitiation.

**kPriLUInit** indicates that the primary LU automatically attempts the reinitiation.

**kSecLUInit** indicates that the secondary LU automatically attempts the reinitiation.

**kEitherLUInit** indicates that either the primary or the secondary LU automatically attempts the reinitiation.

**coSessCrypt** (returned) indicates whether session-level cryptography is supported for this mode. The following values can be returned:

**kFuncNotSupp** indicates that the session-level cryptography is not supported.

**kFuncSupp** indicates that session-level cryptography is supported.

❖ *Note:* At the time of publication, session-level cryptography was not supported.

**coQueueBINDs** (returned) indicates whether or not BINDs sent across this mode can be queued. The following values can be returned:

**kFuncNotSupp** indicates that the BINDs cannot be queued.

**kFuncSupp** indicates that the BINDs can be queued.

**coBlankMode** (returned) indicates whether a null mode name can be sent across the link. The following values can be returned:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that null mode names are not supported. The actual mode name sent across the link is equal to **coModeName**.

**kFuncSupp** indicates that null mode names are supported. The mode name sent across the link is eight space characters. The **kFuncSupp** constant can be provided for only one mode per partner.

**coTermCount** (returned) indicates the number of sessions the local LU is responsible for deactivating as a result of CNOS negotiations.



**coDrainLclLU** (returned) indicates whether or not the local LU is allowed to drain allocation requests.

**coDrainRmtLU** (returned) indicates whether or not the remote LU is allowed to drain allocation requests.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also**      **CODefineMode**



---

## CODisplayRemoteLU

### Summary

The CODisplayRemoteLU routine returns information for a remote LU configured for the specified local LU. For more information on the meaning of values returned by this routine, see the CODEfineRemoteLU routine.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0079	byte	→	coNextRmtLUName
0032	long	⇒	coRmtLUName
003E	long	⇒	coNetName
0042	long	⇒	coNetQual
0046	long	⇒	coCPName
004E	long	⇒	coCNOSALSName
005E	long	⇒	coLUPswd
008E	byte	←	coQueueINITs
0083	byte	←	coParSess
0097	byte	←	coLclSecAcc
0098	byte	←	coRmtSecAcc

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coLclLUName** (supplied) specifies the pointer to a string that contains the local LU name for which the remote LU is defined. The string length must not be greater than the value of the `kMaxName` constant.

**coNextRmtLUName** (supplied) specifies that information for the next configured remote LU should be returned. This parameter is used only if the `coRmtLUName` parameter is specified. The following values are defined:

`kIgnoreParam` indicates that the parameter is not specified.

`kNextEntry` indicates that the information for the next remote LU should be returned.

**coRmtLUName** (supplied/modified) specifies a pointer to space where the name of the remote LU can be specified. The space must be at least as long as the value of the `kMaxName` constant plus 1 byte. If the pointer points to a NULL string, information on the first LU, including its name, is returned. If a name is specified and the `coNextRmtLUName` parameter is set to `kIgnoreParam`, information for the specified LU is returned. If `coNextRmtLUName` is set to the `kNextEntry` constant, information for the next LU, including its name, is returned.

**coNetName** (supplied/modified) specifies a pointer to space where the network name can be returned. The space must be at least as long as the value of the `kMaxTPName` constant plus 1 byte. If this pointer is NIL, the network name is not returned.



**coNetQual** (supplied/modified) specifies a pointer to space where the network qualifier can be returned. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If this pointer is NIL, the name is not returned.

**coCPName** (supplied/modified) specifies a pointer to space where the control point name can be returned. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If this pointer is NIL, the name is not returned.

**coCNOSALName** (supplied/modified) specifies a pointer to space where the name of the adjacent link station used for CNOS negotiation can be returned. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If this pointer is NIL, the name is not returned.

**coLUPswd** (supplied/modified) specifies a pointer to space where the LU-LU password can be returned. The space must be exactly as long as the value of the **kMaxLUPswd** constant plus 1 byte. If this pointer is NIL, the password is not returned.

**coQueueINITs** (returned) indicates whether the system services control point (SSCP) should queue activation requests. The following values can be returned:

**kFuncNotSupp** indicates that the SSCP does not queue session-initiation requests.

**kFuncSupp** indicates that the SSCP does queue session-initiation requests.

**coParSess** (returned) indicates whether or not the remote LU supports parallel sessions. The following values can be returned:

**kFuncNotSupp** indicates that parallel sessions are not supported.

**kFuncSupp** indicates that parallel sessions are supported.

**coLclSecAcc** (returned) indicates the access-security information that the local LU accepts from the remote LU. The following values can be returned:

**kNoSecAcc** indicates that no access-security information is accepted. This value is the default for this parameter.

**kConvSecAcc** indicates that access-security information is accepted but the already-verified indication as set by the allocation request is not accepted.

**kVerifSecAcc** indicates that access-security information is accepted and the already-verified indication as set by the allocation request is accepted.

**coRmtSecAcc** (returned) indicates the access-security information that the remote LU accepts from the local LU. The following values can be returned:

**kNoSecAcc** indicates that no access-security information is accepted. This value is the default for this parameter.

**kConvSecAcc** indicates that access-security information is accepted but the already-verified indication as set by the allocation request is not accepted.

**kVerifSecAcc** indicates that access-security information is accepted and the already-verified indication as set by the allocation request is accepted.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **CODefineRemoteLU**



---

## CODisplaySession

### Summary

The `CODisplaySession` routine returns information about sessions between local LUs and remote LUs. It returns information about a specific session—about the first session for a given mode, or about the next session—thus allowing all sessions for a mode to be displayed.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
0036	long	→	coModeName
007D	byte	→	coNextSessID
006A	long	↔	coSessID
0062	long	←	coConvID
0066	long	←	coProgID
0095	byte	←	coPolarType

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coLclLUName** (supplied) specifies a pointer to a string that contains the name of the local LU. The string length must not be greater than the value of the `kMaxName` constant.

**coRmtLUName** (supplied) specifies a pointer to a string that contains the name of the remote LU. The string length must not be greater than the value of the `kMaxName` constant.

**coModeName** (supplied) specifies a pointer to a string that contains the name of the mode. The string length must not be greater than the value of the `kMaxName` constant.

**coNextSessID** (supplied) specifies that information for the next established session should be returned. This parameter is used only if the `coSessID` parameter is supplied. The following values are defined:

`kIgnoreParam` indicates that the parameter is not specified.

`kNextEntry` indicates that the information for the next established session should be returned.

**coSessID** (supplied/returned) is the ID of the session for which information is returned. If 0 is specified, information for the first session established for the mode is returned, including the session ID.

**coConvID** (returned) gives the ID of the conversation that has allocated this session. If the session is not allocated, 0 is returned.



**coProgID** (returned) gives the transaction program ID of the program that is using the session. If the session is free, 0 is returned.

**coPolarType** (returned) gives the session polarity. The following values can be returned:

`kBidderSess` indicates that the session is a bidder session.

`kFirstSpkrSess` indicates that the session is a first-speaker session.

<b>Result code</b>	<code>appcNoErr</code>	Routine succeeded
	<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
	<code>appcExec</code>	Routine executing; asynchronous request not complete



---

## CODisplayTP

### Summary

The CODisplayTP routine returns information for a transaction program configured for the specified local LU. See the documentation for CODEfineTP for more information on the meaning of values displayed here.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
007B	byte	→	coNextLclProgName
003A	long	⇒	coLclProgName
003E	long	⇒	coNetName
007C	byte	→	coNextUserName
0052	long	⇒	coUserName
005A	long	⇒	coUserProf
008A	word	←	coPIPCount
0088	byte	←	coPIPCheck
0087	byte	←	coPIPReq
0086	byte	←	coFMHDataSupp
0082	byte	←	coLUWSupp
0096	byte	←	coPrivType
0099	byte	←	coSecReq
0093	byte	←	coEnableType
0091	byte	←	coConvType
0090	byte	←	coSyncType
0085	byte	←	coDataMapping

### Description

**coTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**coLclLUName** (supplied) specifies a pointer to a string that contains the local LU name for which the transaction program is defined. The string length must not be greater than the value of the `kMaxName` constant.

**coNextLclProgName** (supplied) specifies that information for the next configured transaction program should be returned. This parameter is used only if the `coLclProgName` parameter is specified. The following values are defined:

`kIgnoreParam` indicates that the parameter is not specified.

`kNextEntry` indicates that the information for the next transaction program should be returned.



**coLclProgName** (supplied/modified) specifies the pointer to space where the name of the transaction program can be returned. The space must be at least as long as the value of the **kMaxTPName** constant plus 1 byte. If the pointer points to a NULL string, information on the first transaction program, including its name, is returned. If a name is specified and the **coNextLclProgName** parameter is set to the **kIgnoreParam** constant, information for the specified transaction program is returned. If **coNextLclProgName** is set to the **kNextEntry** constant, information for the next transaction program, including its name, is returned. If the **kNextEntry** constant is specified and a NULL string is returned, there are no more transaction programs defined for the specified local LU.

**coNetName** (supplied/modified) is a pointer to space where a string that is required only for service transaction programs, such as CNOS and DIA, can be returned. The space must be at least as long as the value of the **kMaxTPName** constant plus 1 byte. The actual network names for such service TPs are found in the TPRM. If this pointer is NIL, the name is not returned.

**coNextUserName** (supplied) specifies that information for the next user ID should be returned. This parameter is used only if the **coUserName** parameter is specified. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kNextEntry** indicates that the information for the next user ID should be returned.

**coUserName** (supplied/modified) specifies the pointer to space where a string that contains the name of a user ID can be returned. The space must be at least as long as the value of the **kMaxSecName** constant plus 1 byte. If the pointer points to a NULL string, the ID and related information for the first user found are returned. If a user ID is supplied and the **coNextUserName** parameter is set to the value of the **kNextEntry** constant, information for the next user is returned. If **coNextUserName** is set to **kIgnoreParam**, information for the specified user is returned. If this pointer is NIL, the user ID is not returned.

**coUserProf** (supplied/modified) specifies the pointer to space where a user profile can be returned. The space must be at least as long as the value of the **kMaxSecName** constant plus 1 byte. If the pointer points to a NULL string, the first profile associated with the returned user ID is returned. If a profile is supplied, the next profile is returned. If this pointer is NIL, the profile is not returned.

**coPIPCount** (returned) indicates the number of PIP subfields that are required on allocation requests that start this transaction program.

**coPIPCheck** (returned) indicates whether allocation requests are rejected if the value specified in the **coPIPCount** parameter does not match the number of PIP subfields supplied on the request. The following values can be returned:

**kFuncNotSupp** indicates that requests are not rejected.

**kFuncSupp** indicates that request are rejected.

**coPIPReq** (returned) indicates whether or not PIP data is required on allocation requests designating this transaction program. The following values can be returned:

**kFuncNotSupp** indicates that no PIP data is allowed on allocation requests.

**kFuncSupp** indicates that PIP data is allowed on allocation requests.



**coFMHDataSupp** (returned) indicates whether or not FMH data can be received and sent by this transaction program. The following values can be returned:

**kFuncNotSupp** indicates that no PIP data is allowed on allocation requests.

**kFuncSupp** indicates that PIP data is allowed on allocation requests.

**coLUWSupp** (returned) indicates whether or not the transaction program assigns logical-unit-of-work IDs for each transaction executed by this program. The following values can be returned:

**kFuncNotSupp** indicates that logical-unit-of-work IDs are not assigned.

**kFuncSupp** indicates that logical-unit-of-work IDs are assigned.

**coPrivType** (returned) indicates the type of control operator routines that this transaction program can execute. The following values can be returned (any values except **kNoPriv** can be combined in a logical OR operation):

**kNoPriv** indicates that the program is not allowed to execute routines that require a privilege to do so.

**kCNOSPriv** indicates that the program is allowed to execute CNOS routines.

**kSessCtlPriv** indicates that the program is allowed to execute session-control routines.

**kDefinePriv** indicates that the program is allowed to execute definition routines.

**kDisplayPriv** indicates that the program is allowed to execute display routines.

**kSvcTPPriv** indicates that the program is allowed to execute an allocation request that specifies an SNA service transaction program.

**coSecReq** (returned) indicates the security verification allowed on an allocation request to start the program. The following values can be returned:

**kNoSecReq** indicates that no security is required.

**kConvSecReq** indicates that conversation-level security is required.

**kProfSecReq** indicates that resource-access security is required at the profile level.

**kUserNameSecReq** indicates that resource-access security is required at the user ID level.

**kBothSecReq** indicates that resource-access security is required at both the profile and user ID levels.

**coEnableType** (returned) indicates the response that the LU makes when an allocation request for this transaction program is received. The following values can be returned:

**kEnableTP** indicates that the transaction program is available.

**kDisableTempTP** indicates that the transaction program is temporarily unavailable.

**kDisablePermTP** indicates that the transaction is permanently unavailable.



**coConvType** (returned) indicates the conversation type allowed on an allocation request to start the program. The following values can be returned:

**kMappedTPConv** specifies that the allocation request must specify a mapped conversation.

**kBasicTPConv** specifies that the allocation request must specify a basic conversation.

**kEitherTPConv** specifies that the allocation request can specify either a basic or mapped conversation.

**coSyncType** (returned) indicates the synchronization allowed on an allocation request to start the program. The following values can be returned (any values can be combined in a logical OR operation) :

**kNoTPSync** indicates that the allocation request can specify no confirmation processing.

**kConfirmTPSync** indicates that the allocation request can specify confirmation processing.

**kSyncPtTPSync** indicates that the allocation request can specify sync-point processing.

❖ *Note:* At the time of publication, sync-point services were not supported.

**coDataMapping** (returned) indicates whether or not the transaction program is provided with data mapping support. The following values can be returned:

**kFuncNotSupp** indicates that data mapping is not supported.

**kFuncSupp** indicates that data mapping is supported.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **CODefineTP**



---

---

## Summary of the MacAPPC Control Operator Driver

This section provides a summary of the constants, data structures, and routines for use with the MacAPPC Control Operator Driver.

---

### Constants

The following constants are available for use with the MacAPPC Control Operator Driver.

```
{ coRespType values }

kSrcResp =          0;
kTgtResp =          1;

{ coDeactType values }

kNormalDeact =       0;
kCleanupDeact =      1;

{ coSyncType values }

kConfirmModeSync =   1;
kSyncPtModeSync =    2;          { not supported }

kNoTPSync =          1;
kConfirmTPSync =     2;
kSyncPtTPSync =      4;          { not supported }

{ coConvType values }

kMappedTPConv =      0;
kBasicTPConv =       1;
kEitherTPConv =      2;

{ coEnableType values }

kEnableTP =          0;
kDisableTempTP =     1;
kDisablePermTP =     2;

{ coPolarType values }

kBidderSess =        0;
kFirstSpkrSess =     1;
```



{ coPrivType values }

kNoPriv = 0;  
kCNOSPriv = 1;  
kSessCtlPriv = 2;  
kDefinePriv = 4;  
kDisplayPriv = 8;  
kSvcTPPriv = 16;

{ coLclSecAccType and coRmtSecAccType values }

kNoSecAcc = 0;  
kConvSecAcc = 1;  
kVerifSecAcc = 2;

{ coSecReq values }

kNoSecReq = 0;  
kConvSecReq = 1;  
kProfSecReq = 2;  
kUserSecReq = 3;  
kBothSecReq = 4;

{ coReinitType values }

kOperInit = 0;  
kPriLUInit = 1;  
kSecLUInit = 2;  
kEitherLUInit = 3;



## Data types

The following data types are available for use with the MacAPPC Control Operator Driver.

```
coParam:
(
coTPCBPtr      : Ptr;          { TPCB pointer }
coCVCBPtr      : Ptr;          { CVCB pointer }
coLclLUName    : StringPtr;    { local LU name pointer }
coRmtLUName    : StringPtr;    { remote LU name pointer }
coModeName     : StringPtr;    { mode name pointer }
coLclProgName  : StringPtr;    { local TP name pointer }
coNetName      : StringPtr;    { network name pointer }
coNetQual      : StringPtr;    { network qualifier pointer }
coCPName       : StringPtr;    { control point name pointer }
coALSName      : StringPtr;    { adjacent link station name pointer }
coCNOSALSName  : StringPtr;    { CNOS station name pointer }
coUserName     : StringPtr;    { user name pointer }
coUserPswd     : StringPtr;    { user password pointer }
coUserProf     : StringPtr;    { user profile pointer }
coLUPswd       : StringPtr;    { LU-LU password pointer }
coConvID       : LONGINT;      { conversation ID }
coProgID       : LONGINT;      { transaction program ID }
coSessID       : LONGINT;      { session ID }
coLclLUID      : SignedByte;   { local LU ID }
coWaitTime     : INTEGER;      { wait time in seconds }
coDrainSrc     : Boolean;       { reset drain source }
coDrainTgt     : Boolean;       { reset drain target }
coForceRst     : Boolean;       { force reset }
coDrainLclLU   : Boolean;       { drain local LU }
coDrainRmtLU   : Boolean;       { drain remote LU }
coLUActive     : Boolean;       { LU activation status }
coNextLclLUName : SignedByte;  { display next local LU }
coNextRmtLUName : SignedByte;  { display next remote LU }
coNextModeName : SignedByte;   { display next mode }
coNextLclProgName : SignedByte; { display next local TP }
coNextUserName : SignedByte;   { display next user }
coNextSessID   : SignedByte;   { display next session ID }
coNetNameOp    : SignedByte;   { network name operation }
coNetQualOp    : SignedByte;   { network qualifier operation }
coLUPswdOp     : SignedByte;   { password operation }
coSecOp        : SignedByte;   { security operation }
coLUWSupp      : SignedByte;   { LUW support }
coParSess      : SignedByte;   { parallel session support }
coQueueBINDs   : SignedByte;   { queue BINDs }
coDataMapping  : SignedByte;   { data mapping support }
coFMHDataSupp  : SignedByte;   { FMH data support }
coPIPReq       : SignedByte;   { PIP required }
coPIPCheck     : SignedByte;   { check PIP count }
coPIPCount     : INTEGER;      { PIP count }
coSessCrypt    : SignedByte;   { session-level cryptography }
coBlankMode    : SignedByte;   { blank mode option }
```



```

coQueueINITS      : SignedByte; { initiate type }
coConvSecType     : SignedByte; { conversation-level security }
coSyncType        : SignedByte; { synchronization level }
coConvType        : SignedByte; { conversation type }
coDeactType       : SignedByte; { deactivate type }
coEnableType      : SignedByte; { enable status type }
coRespType        : SignedByte; { deactivate responsibility }
coPolarType       : SignedByte; { session polarity }
coPrivType        : SignedByte; { privilege }
coLclSecAcc       : SignedByte; { local LU security acceptance }
coRmtSecAcc       : SignedByte; { remote LU security acceptance }
coSecReq          : SignedByte; { security required }
coReinitType      : SignedByte; { single session reinitiation }
coTermCount       : INTEGER; { termination count }
coMaxTP           : INTEGER; { maximum attached TPs }
coSendPacing      : INTEGER; { send pacing window }
coRcvPacing       : INTEGER; { receive pacing window }
coMaxRUHiBound    : INTEGER; { maximum RU upper bound }
coMaxRULoBound    : INTEGER; { maximum RU lower bound }
coDefLUMaxSess    : INTEGER; { defined max LU sessions }
coActLUSess       : INTEGER; { active LU sessions }
coDefMaxSess      : INTEGER; { defined max sessions }
coCurMaxSess      : INTEGER; { current max sessions }
coActSess         : INTEGER; { active sessions }
coDefMinFirstSpkrs : INTEGER; { defined min first speakers }
coCurMinFirstSpkrs : INTEGER; { current min first speakers }
coActFirstSpkrs   : INTEGER; { active first speakers }
coDefMinBdrs      : INTEGER; { defined min bidders }
coCurMinBdrs      : INTEGER; { current min bidders }
coActBdrs         : INTEGER; { active bidders }
coDefPBFfirstSpkrs : INTEGER; { prebound first speakers }
);

```



---

## CNOS routines

The following change-number-of-sessions routines are available for use with the MacAPPC Control Operator Driver.

### COChangeSessionLimit

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002A	long	→	coCVCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName
00AE	word	→	coCurMaxSess
00B4	word	→	coCurMinFirstSpkrs
00BA	word	→	coCurMinBdrs
0094	byte	→	coRespType

### COInitializeSessionLimit

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002A	long	→	coCVCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName
00AE	word	→	coCurMaxSess
00B4	word	→	coCurMinFirstSpkrs
00BA	word	→	coCurMinBdrs

### COProcessSessionLimit

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002A	long	→	coCVCBPtr
0032	long	⇒	coRmtLUName
0036	long	⇒	coModeName



## COResetSessionLimit

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002A	long	→	coCVCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName
0094	byte	→	coRespType
0072	byte	→	coDrainSrc
0073	byte	→	coDrainTgt
0074	byte	→	coForceRst



---

## Session control routines

The following session control routines are available for use with the MacAPPC Control Operator Driver.

### COActivateSession

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName

### CODeactivateSession

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
0032	long	→	coRmtLUName
0036	long	→	coModeName
006A	long	→	coSessID
0092	byte	→	coDeactType



---

## LU definition routines

The following LU definition routines are available for use with the MacAPPC Control Operator Driver.

### CODefineLocalLU

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
007E	byte	→	coNetNameOp
003E	long	→	coNetName
007F	byte	→	coNetQualOp
0042	long	→	coNetQual
0081	byte	→	coSecOp
0052	long	→	coUserName
0056	long	→	coUserPswd
005A	long	→	coUserProf
0070	word	→	coWaitTime
00A8	word	→	coDefLUMaxSess
009E	word	→	coMaxTP
006E	byte	→	coLclLUID
008F	byte	→	coConvSecType

### CODefineMode

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
0036	long	→	coModeName
004A	long	→	coALSName
00A0	word	→	coSendPacing
00A2	word	→	coRcvPacing
00AC	word	→	coDefMaxSess
00B2	word	→	coDefMinFirstSpkrs
00BE	word	→	coDefPBFFirstSpkrs
00A4	word	→	coMaxRUHiBound
00A6	word	→	coMaxRULoBound
0090	byte	→	coSyncType
009A	byte	→	coReinitType
008C	byte	→	coSessCrypt
0084	byte	→	coQueueBINDs
008D	byte	→	coBlankMode



## CODefineRemoteLU

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
007E	byte	→	coNetNameOp
003E	long	→	coNetName
007F	byte	→	coNetQualOp
0042	long	→	coNetQual
0046	long	→	coCPName
004E	long	→	coCNOSALSName
0080	byte	→	coLUPswdOp
005E	long	→	coLUPswd
008E	byte	→	CoQueueINITS
0083	byte	→	coParSess
0097	byte	→	coLclSecAcc

## CODefineTP

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
003A	long	→	coLclProgName
007E	byte	→	coNetNameOp
003E	long	→	coNetName
0052	long	→	coUserName
005A	long	→	coUserProf
008A	word	→	coPIPCount
0088	byte	→	coPIPCheck
0087	byte	→	coPIPReq
0086	byte	→	coFMHDataSupp
0082	byte	→	coLUWSupp
0096	byte	→	coPrivType
0099	byte	→	coSecReq
0081	byte	→	coSecOp
0093	byte	→	coEnableType
0091	byte	→	coConvType
0090	byte	→	coSyncType
0085	byte	→	coDataMapping



## CODelete

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
003A	long	→	coLclProgName
0036	long	→	coModeName

## CODisplayLocalLU

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
0078	byte	→	coNextLclLUName
002E	long	⇒	coLclLUName
003E	long	⇒	coNetName
0042	long	⇒	coNetQual
007C	byte	→	coNextUserName
0052	long	⇒	coUserName
0056	long	⇒	coUserPswd
005A	long	⇒	coUserProf
0070	word	←	coWaitTime
00A8	word	←	coDefLUMaxSess
00AA	word	←	coActLUSess
009E	word	←	coMaxTP
006E	byte	←	coLclLUID
008F	byte	←	coConvSecType
0077	byte	←	coLUActive

## CODisplayMode

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
007A	byte	→	coNextModeName
0036	long	⇒	coModeName
004A	long	⇒	coALSName



00A0	word	←	coSendPacing
00A2	word	←	coRcvPacing
00A4	word	←	coMaxRUHiBound
00A6	word	←	coMaxRULoBound
00AC	word	←	coDefMaxSess
00B2	word	←	coDefMinFirstSpkrs
00BE	word	←	coDefPBFirstSpkrs
00B8	word	←	coDefMinBdrs
00AE	word	←	coCurMaxSess
00B4	word	←	coCurMinFirstSpkrs
00BA	word	←	coCurMinBdrs
00B0	word	←	coActSess
00B6	word	←	coActFirstSpkrs
00BC	word	←	coActBdrs
0090	byte	←	coSyncType
009A	byte	←	coReinitType
008C	byte	←	coSessCrypt
0084	byte	←	coQueueBINDs
008D	byte	←	coBlankMode
009C	byte	←	coTermCount
0075	byte	←	coDrainLclLU
0076	byte	←	coDrainRmtLU

#### **CODisplayRemoteLU**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0079	byte	→	coNextRmtLUName
0032	long	⇒	coRmtLUName
003E	long	⇒	coNetName
0042	long	⇒	coNetQual
0046	long	⇒	coCPName
004E	long	⇒	coCNOSALSName
005E	long	⇒	coLUPswd
008E	byte	←	CoQueueINITs
0083	byte	←	coParSess
0097	byte	←	coLclSecAcc
0098	byte	←	coRmtSecAcc

#### **CODisplaySession**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult

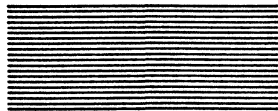


0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
0032	long	→	coRmtLUName
0036	long	→	coModeName
007D	byte	→	coNextSessID
006A	long	↔	coSessID
0062	long	←	coConvID
0066	long	←	coProgID
0095	byte	←	coPolarType

# **CODisplayTP**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	coTPCBPtr
002E	long	→	coLclLUName
007B	byte	→	coNextLclProgName
003A	long	⇒	coLclProgName
003E	long	⇒	coNetName
007C	byte	→	coNextUserName
0052	long	⇒	coUserName
005A	long	⇒	coUserProf
008A	word	←	coPIPCount
0088	byte	←	coPIPCheck
0087	byte	←	coPIPReq
0086	byte	←	coFMHDataSupp
0082	byte	←	coLUWSupp
0096	byte	←	coPrivType
0099	byte	←	coSecReq
0093	byte	←	coEnableType
0091	byte	←	coConvType
0090	byte	←	coSyncType
0085	byte	←	coDataMapping





## **Chapter 6**

# **MacAPPC Node Operator Driver**



This chapter describes the MacAPPC Node Operator Driver (.NO62), explains how to use the driver, and provides a detailed guide to the programmatic interface for executing each Node Operator Driver routine. For quick reference, a section at the end of the chapter summarizes the data structures, constants, and routine parameters.

Unlike conversation and control operator functions, the node operator functions are not defined by the SNA architecture; it is understood that they must exist, but they are implementation specific.

---

---

## Using the MacAPPC Node Operator Driver

Node operator routines permit a node operator to define and control certain components of a PU 2.1 node. They activate and deactivate the node, the lines, the LUs, and the stations; in addition, they define and display the node, the lines, the control points, the stations, and the message queues.

To make the programmer's task easier, MacAPPC provides a node operator program—the Administration program, described in Chapter 12—that handles all of the node operator functions. By using the Administration program, you do not need to access the Node Operator Driver in the programs that you write; however, if you choose to do so, the node operator routines are fully described in this chapter.

---

---

## MacAPPC node operator routines

This section describes the MacAPPC node operator routines, which define, display, and control node components. The following node components are defined:

- ☐ Node
- ☐ Line
- ☐ Control point
- ☐ Station

The routines are divided into the following categories:

- ☐ Node control routines, which control node components
- ☐ Node message routines, which define and display node message queues and messages
- ☐ Node definition routines, which define and display node components

---

---

## Node operator node control routines

This section describes the MacAPPC node operator node control routines. These routines activate and deactivate node components.



---

## NOActivateLine

### Summary

The `NOActivateLine` routine activates the line for use by the MacAPPC server. This routine completely activates the connection when the line is non-switched, point-to-point. If the line is switched or multipoint, the `NOActivateStation` routine must also be executed to complete the connection.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
004E	long	←	noCorrID

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noLineName** (supplied) specifies a pointer to a string that contains the name of the line to be activated. The string length must not be greater than the value of the `kMaxName` constant.

**noCorrID** (returned) provides the correlation number of the request that subsequent `NODisplayMessage` routines can use. If the request is not accepted, the value 0 is returned.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`NODEactivateLine`, `NOActivateStation`



---

## NOActivateLU

**Summary** The NOActivateLU routine locally activates an LU for use by the MacAPPC server. It must be successfully completed before any transaction program can attach itself to that LU.

<b>Parameters</b>	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0026	long	→	noTPCBPtr
	002A	long	→	noLclLUName
	004E	long	←	noCorrID

**Description** **noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noLclLUName** (supplied) specifies a pointer to a string that contains the name of the local LU to be activated. The string length must not be greater than the value of the **kMaxName** constant.

**noCorrID** (returned) provides the correlation number of the request that subsequent **NODisplayMessage** routines can use. If the request is not accepted, the value 0 is returned.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** NODeactivateLU



---

## NOActivateNode

### Summary

The NOActivateNode routine locally activates a MacAPPC server. This is a prerequisite for all other activity of the MacAPPC server.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
004E	long	←	noCorrID

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noCorrID** (returned) provides the correlation number of the request that subsequent NODisplayMessage routines can use. If the request is not accepted, the value 0 is returned.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

NODEactivateNode



---

## NOActivateStation

### Summary

The NOActivateStation routine instructs the MacAPPC server to execute the proper commands to enable the connection to a particular remote node. It is used to connect to a particular station on a multipoint line or to enable dial-in or dial-out on a switched connection. The NOActivateLine routine must also be executed for the line named in this routine.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
002E	long	→	noALSName
005A	byte	→	noDialType
004E	long	←	noCorrID

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noALSName** (supplied) specifies a pointer to a string that contains the name of the station to be activated. The string length must not be greater than the value of the kMaxName constant.

**noDialType** (supplied) has two possible values:

**kConnectDial** specifies an attempt to connect for a multipoint line. For a switched line, it indicates an attempt to dial-out.

**kDialInOnDial** specifies that the dial-in capability of the line is enabled.

**noCorrID** (returned) provides the correlation number of the request that subsequent NODisplayMessage routines can use. If the request is not accepted, the value 0 is returned.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

NODEactivateStation, NOActivateLine



---

## NODEactivateLine

### Summary

The NODEactivateLine routine deactivates the named line from use by the MacAPPC server. NODEactivateLine completely deactivates the line and cancels all outstanding requests for that line regardless of the line type.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
004E	long	←	noCorrID

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noLineName** (supplied) specifies a pointer to a string that contains the name of the line to be deactivated. The string length must not be greater than the value of the kMaxName constant.

**noCorrID** (returned) provides the correlation number of the request that subsequent NODisplayMessage routines can use. If the request is not accepted, the value 0 is returned.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

NOActivateLine



---

## NODEactivateLU

### Summary

The NODEactivateLU routine terminates the use of the named logical unit. Any sessions that have been activated are terminated. After this routine is successfully completed, no transaction program can attach itself to this LU until an NOActivateLU routine is executed.

<b>Parameters</b>	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0026	long	→	noTPCBPtr
	002A	long	→	noLclLUName
	004E	long	←	noCorrID

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noLclLUName** (supplied) specifies a pointer to a string that contains the name of the LU to be deactivated. The string length must not be greater than the value of the kMaxName constant.

**noCorrID** (returned) provides the correlation number of the request that subsequent NODisplayMessage routines can use. If the request is not accepted, the value 0 is returned.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

NOActivateLU



---

## NODEactivateNode

**Summary** The NODEactivateNode routine locally deactivates a MacAPPC server. All MacAPPC server activity is cancelled.

**Parameters**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0059	byte	→	noStopSrvr
004E	long	←	noCorrID

**Description** **noTPCBPtr** (supplied) is a pointer to an existing Transaction Program Control Block (TPCB).

**noStopSrvr** (supplied) specifies one of two possible values: TRUE deactivates the MacAPPC server after the NODEactivateNode routine has completed; FALSE indicates that the MacAPPC server continues to run after the NODEactivateNode routine has been completed.

**noCorrID** (returned) provides the correlation number of the request that subsequent NODisplayMessage routines can use. If the request is not accepted, the value 0 is returned.

**Result code**

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

**See also** NOActivateNode



---

## NODEactivateStation

### Summary

The NODEactivateStation routine instructs the MacAPPC server to execute the proper commands to disable the link connection to a particular remote node. It is used to disconnect a particular station on a multipoint line or to disable dial-in or dial-out on a switched connection. To completely terminate the line, the NODEactivateLine routine must also be executed for the line named in this routine.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
002E	long	→	noALSName
005A	byte	→	noDialType
004E	long	←	noCorrID

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noALSName** (supplied) specifies a pointer to a string that contains the name of the station to be deactivated. The string length must not be greater than the value of the kMaxName constant.

**noDialType** (supplied) specifies the dial type. The following values are defined:

kDisconnectDial indicates that the connection to the named station is broken. This parameter is ignored if the line is not switched.

kDialInOffDial indicates that the dial-in is disabled. The current connection is not dropped.

**noCorrID** (returned) provides the correlation number of the request that subsequent NODisplayMessage routines can use. If the request is not accepted, the value 0 is returned.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

NOActivateStation, NODEactivateLine



---

---

## **Node operator node message routines**

This section describes the MacAPPC Node Operator node message routines. These routines define and display internal MacAPPC server message queues and messages.



---

## NODefineMessageQueue

### Summary

The `NODefineMessageQueue` routine sends a request to the MacAPPC server to change the attributes of a message queue. The queue must already exist.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
003A	long	→	noQueueName
005B	byte	→	noQueueEnable
005C	byte	→	noQueueClass
005D	byte	→	noQueueType
005E	byte	→	noQueueSev

### Description

**noTPCBPtr** (supplied) is a pointer to an existing Transaction Program Control Block (TPCB).

**noQueueName** (supplied) specifies a pointer to a string that contains the name of an existing message queue. The string length must not be greater than the value of the `kMaxName` constant. At the time of publication, valid names are NOOP and LOG.

**noQueueEnable** (supplied) specifies one of two possible values: FALSE indicates that the queue should be disabled; TRUE indicates that the queue should be enabled.

**noQueueClass** (supplied) is the class of messages to be written to this message queue. The following values are defined (any values except `kNoChangeQClass` can be combined in a logical OR operation):

`kNoChangeQClass` indicates no change.

`kNodeOperMsgsQClass` indicates node-operator messages.

`kLogMsgsQClass` indicates logging messages.

`kDevelMsgsQClass` indicates development messages.

`kTraceMsgsQClass` indicates trace messages.

**noQueueType** (supplied) is the type of messages to be written to this message queue. The following values are defined (any values except `kNoChangeQType` can be combined in a logical OR operation):

`kNoChangeQType` indicates no change.

`kInfoMsgsQType` indicates informational messages.

`kNotifMsgsQType` indicates notification messages.

`kErrorMsgsQType` indicates error messages.

`kDiagMsgsQType` indicates diagnostic messages.



**noQueueSev** (supplied) is the severity level of messages to be written to this message queue. The following values are defined:

**kNoChangeQSev** indicates no change to the current level.

**kDevelMsgsQSev** indicates development messages.

**kLowLevelInfoMsgsQSev** indicates low-level informational messages.

**kNormalInfoMsgsQSev** indicates normal informational messages.

**kErrorMsgsQSev** indicates error messages.

**kProgErrorsQSev** indicates programming errors.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **NODisplayMessageQueue**



---

## NODisplayMessage

**Summary** The NODisplayMessage routine retrieves a message from the named message queue.

**Parameters**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
003A	long	→	noQueueName
0050	long	⇒	noDataPtr
0054	word	→	noDataSize
005F	byte	→	noWaitForMsg
004E	long	→	noCorrID

**Description** **noTPCBPtr** (supplied) is a pointer to an existing Transaction Program Control Block (TPCB).

**noQueueName** (supplied) specifies a pointer to a string that contains the name of an existing message queue. The string length must not be greater than the value of the **kMaxName** constant. At the time of publication, valid names are NOOP and LOG.

**noDataPtr** (supplied/modified) specifies a pointer to space where this routine is to copy the message. The size of the space is specified by the **noDataSize** parameter. If this pointer is NIL and a message is found, the result codes indicate that a message exists, but the message is not retrieved.

**noDataSize** (supplied) is the maximum message length that can be copied. If the message is longer than this length, it is truncated.

**noWaitForMsg** (supplied) specifies whether or not the transaction program is to wait for a message if none are available. TRUE means that the transaction program is to wait the amount of time specified in the **tpwaitTime** parameter of the **TPAttach** routine; FALSE means that the transaction program is not to wait.

**noCorrID** (supplied) provides the correlation value that can be used to match a specific reply. If it is not provided, the first message is returned.

**Result code**

appcNoErr	Routine succeeded
appcFail	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
appcExec	Routine executing; asynchronous request not complete

**See also** NODefineMessageQueue, NODisplayMessageQueue



---

## NODisplayMessageQueue

### Summary

The `NODisplayMessageQueue` routine sends a request to the MacAPPC server to display the attributes of a message queue. The queue must already exist.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
003A	long	→	noQueueName
005B	byte	←	noQueueEnable
005C	byte	←	noQueueClass
005D	byte	←	noQueueType
005E	byte	←	noQueueSev

### Description

**noTPCBPtr** (supplied) is a pointer to an existing Transaction Program Control Block (TPCB).

**noQueueName** (supplied) specifies a pointer to a string that contains the name of an existing message queue. The string length must not be greater than the value of the `kMaxName` constant. At the time of publication, valid names are NOOP and LOG.

**noQueueEnable** (returned) specifies one of two possible values: FALSE indicates the queue is disabled; TRUE indicates the queue is enabled.

**noQueueClass** (returned) is the class of messages to be written to this message queue. The following values can be returned (any values can be combined in a logical OR operation):

`kNodeOperMsgsQClass` indicates node-operator messages.

`kLogMsgsQClass` indicates logging messages.

`kDevelMsgsQClass` indicates development messages.

`kTraceMsgsQClass` indicates trace messages.

**noQueueType** (returned) is the type of message to be written to this queue. The following values can be returned (any values can be combined in a logical OR operation):

`kInfoMsgsQType` indicates informational messages.

`kNotifMsgsQType` indicates notification messages.

`kErrorMsgsQType` indicates error messages.

`kDiagMsgsQType` indicates diagnostic messages.



**noQueueSev** (returned) is the severity level of messages to be written to this message queue. The following values can be returned:

**kDevelMsgsQSev** indicates development messages.

**kLowLevelInfoMsgsQSev** indicates low-level informational messages.

**kNormalInfoMsgsQSev** indicates normal informational messages.

**kErrorMsgsQSev** indicates error messages.

**kProgErrorsQSev** indicates programming errors.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **NODefineMessageQueue**



---

---

## **Node operator node definition routines**

This section describes the MacAPPC node operator node definition routines. These routines define, display, and delete node components.



---

## NODefineCP

### Summary

The NODefineCP routine defines the addressing information for the control point at the remote node. There are two types of control points: hosts and peers. Host nodes (PU type 4 or 5) are known by their CPU IDs. Peer nodes (PU type 2.1) are known by their exchange IDs. You must specify one, but not both. This routine is used to initialize and modify these parameters.

This routine is used to initialize and modify operating parameters that are located in the new or existing CP. The first time it is executed, this routine initializes the CP definition with default values and updates it with the specified operating parameters. On subsequent executions this routine updates the CP definition with the data supplied for the specified operating parameters. If a parameter is not specified, the value currently defined for that parameter remains unchanged. This routine can be executed by any transaction program that has the privilege to execute definition routines.

Parameters	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0026	long	→	noTPCBPtr
	0036	long	→	noCPName
	0046	long	→	noExchID
	004A	long	→	noCPUID

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noCPName** (supplied) specifies a pointer to a string that contains the name of the control point. The string length must not be greater than the value of the `kMaxName` constant. The string `LOCAL` is a reserved name designating intra-node flows and must not be used as the name of a control point. The `noCPName` parameter is used in the `NODefineStation` and `NODefineRemoteLU` routines to specify the location of the station or remote LU.

**noExchID** (supplied) specifies a pointer to a string that contains the XID of the remote node if the node is a peer; specifies a NIL pointer if the node is a host. The string must be exactly as long as the value of the `kMaxExchID` constant, and the characters in the string must be hexadecimal.

The string is used in XID exchange at link establishment. In IBM implementations, the first three characters are set based on the product type (03A for the Displaywriter, 03E for the System/36, and so on). The next five characters are user-configurable to give unique exchange IDs throughout the network. If the partner is an IBM product, check the relevant IBM manuals for restrictions on the exchange ID value. This value must match the value sent by the remote node in bytes 2-5 of the XID.



**noCPUID** (supplied) specifies a pointer to a string that contains the CPU ID of the remote node if the node is a host; specify a NIL pointer if the node is a peer. The string must be exactly as long as the value of the **kMaxCPUID** constant, and the characters in the string must be hexadecimal. The first two characters of the string (representing the first byte of the CPU ID) are the PU type (usually 05 for PU type 5). The remaining ten characters of the string (representing the final 5 bytes) are an implementation-dependent binary identifier.

On a VTAM host, the 5 bytes are set to the subarea identifier of the host, specified by the SSCPID keyword in the VTAM ATCSTR definition. This value must match the value sent by the system services control point (SSCP) in bytes 3–8 of the Activate Physical Unit (ACTPU) request.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also**      **NODisplayCP**, **NODefineStation**



---

## NODefineLine

### Summary

The NODefineLine routine defines the line name, line type, and characteristics of a line.

This routine is used to initialize and modify operating parameters that are located in the new or existing line. The first time it is executed, this routine initializes the line definition with default values and updates it with the specified operating parameters. On subsequent executions this routine updates the line definition with the data supplied for the specified operating parameters. If a parameter is not specified, the value currently defined for that parameter remains unchanged. This routine can be executed by any transaction program that has the privilege to execute definition routines.

Parameters	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0026	long	→	noTPCBPtr
	0032	long	→	noLineName
	0068	long	→	noLinePtr
	0066	byte	→	noLineType
	----	byte	→	sdlcLineNum
	----	byte	→	sdlcRoleType
	----	byte	→	sdlcConnType
	----	word	→	sdlcMaxBTU
	----	word	→	sdlcLineSpeed
	----	word	→	sdlcMaxRetry
	----	word	→	sdlcIdleTime
	----	word	→	sdlcNPRcvTime
	----	word	→	sdlcMaxIFrame
	----	byte	→	sdlcNRZIType
	----	byte	→	sdlcDuplexType

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noLineName** (supplied) specifies a pointer to a string that contains the name of the line being defined or modified. The string length must not be greater than the value of the **kMaxName** constant.

**noLinePtr** (supplied) specifies a pointer to a line record (APPCLineRec).

**noLineType** (supplied) specifies the type of line. The following values are defined:

**kSDLCLine** indicates an SDLC line type.

♦ *Note:* At the time of publication, this is the only line type supported.



**sdlcLineNum** (supplied) specifies the SDLC line number. The following values are defined:

- kIgnoreParam indicates that the parameter is not specified.
- kSDLCLine1 indicates SDLC line 1. This value is the default for this parameter.
- kSDLCLine2 indicates SDLC line 2.
- kSDLCLine3 indicates SDLC line 3.
- kSDLCLine4 indicates SDLC line 4.

**sdlcRoleType** (supplied) specifies the SDLC role. The following values are defined:

- kIgnoreParam indicates that the parameter is not specified.
- kSDLCPrimary indicates that the SDLC role is primary.
- kSDLCSecondary indicates that the SDLC role is secondary.
- kSDLCNegotiable indicates that the SDLC role is negotiable. This value is the default for this parameter.

**sdlcConnType** (supplied) specifies the SDLC connection type. The following values are defined:

- kIgnoreParam indicates that the parameter is not specified.
- kSDLCLeased indicates that the connection is leased. This value is the default for this parameter.
- kSDLCMultiPoint indicates that the connection is multipoint. If kSDLCMultiPoint is selected, the **sdlcRoleType** parameter must be set to the kSDLCPrimary constant.
- kSDLCSwitched indicates that the connection is switched. If kSDLCSwitched is specified, the **sdlcDuplexType** parameter must be set to kSDLCHalfDuplex.

**sdlcMaxBTU** (supplied) specifies the maximum basic transmission unit (BTU) length that can be received on this line. This value is exchanged with the partner at link initiation. The valid range is 128 to 265. The default value is 265. The kIgnoreParam constant indicates that the parameter is not specified.

**sdlcLineSpeed** (supplied) specifies the SDLC line speed. The following values are defined:

- kIgnoreParam indicates that the parameter is not specified.
- kSDLC300 indicates a line speed of 300 bps.
- kSDLC1200 indicates a line speed of 1200 bps.
- kSDLC2400 indicates a line speed of 2400 bps.
- kSDLC4800 indicates a line speed of 4800 bps.
- kSDLC9600 indicates a line speed of 9600 bps. This value is the default for this parameter.
- kSDLC19200 indicates a line speed of 19200 bps.



**sdlcMaxRetry** (supplied) specifies the maximum number of times a frame is retransmitted after SDLC procedures have detected a discrepancy. When this number is exceeded, SDLC reports the problem to a higher level of SNA for resolution. The valid range is 1 to 30. The default value is 3. Specify 0 to indicate there is no maximum number of retries. The **kIgnoreParam** constant indicates that the parameter is not specified.

**sdlcIdleTime** (supplied) specifies the amount of time that can elapse without a response before a primary link station will initiate recovery action. The valid range in milliseconds is 100 to 10000. The default value for this parameter is 800 milliseconds. The **kIgnoreParam** constant indicates that the parameter is not specified.

**sdlcNPRcvTime** (supplied) specifies the amount of time that can elapse before the primary link station reports to a higher level of SNA that a nonproductive receive condition exists. The valid range in milliseconds is 1000 to 30000. The default value for this parameter is 10000 milliseconds. The **kIgnoreParam** constant indicates that the parameter is not specified.

**sdlcMaxIFrame** (supplied) specifies the maximum number of I-frames that can be sent before polling resumes. The valid range is 1 to 7. The default value for this parameter is 7. The **kIgnoreParam** constant indicates that the parameter is not specified.

**sdlcNRZIType** (supplied) specifies the transmission encoding method to be used when sending signals over this link. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kSDLCNRZ** indicates the use of nonreturn-on-zero (NRZ) encoding. This value is the default for this parameter.

**kSDLCNRZI** indicates the use of nonreturn-on-zero inverted (NRZI) encoding.

**sdlcDuplexType** (supplied) specifies whether or not data can be sent in one or both directions without turning the line around. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kSDLCHalfDuplex** indicates half-duplex transmissions. Data can be sent in only one direction and then the line must be turned around.

**kSDLCFullDuplex** indicates full-duplex transmissions. Data can be sent and received without turning the line around. This value is the default for this parameter.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **NODisplayLine**



---

## NODefineNode

### Summary

The NODefineNode routine initializes and modifies the system-wide operating parameters for an instance of the MacAPPC Server.

This routine is used to initialize and modify operating parameters that are located in the new or existing node. The first time it is executed, this routine initializes the node definition with default values and updates it with the specified operating parameters. On subsequent executions this routine updates the node definition with the data supplied for the specified operating parameters. If a parameter is not specified, the value currently defined for that parameter remains unchanged. This routine can be executed by any transaction program that has the privilege to execute definition routines.

Parameters	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0026	long	→	noTPCBPtr
	0046	long	→	noExchID
	0058	byte	→	noAccessType
	0056	word	→	noMonTimer
	0063	byte	→	noNodeMsgs
	0064	byte	→	noLogMsgs

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noExchID** (supplied) specifies a pointer to a string that contains the XID of the local node. The string must be exactly as long as the value of the **kMaxExchID** constant, and the characters in the string must be hexadecimal.

The string is used in XID exchange at link establishment. If the partner is an IBM product, check the relevant IBM manual for restrictions on exchange ID values. The default value is 02200001 (equivalent to an IBM System/38). If the pointer is NIL, the current value is not changed. This value will be sent in bytes 2–5 of the XID.

**noAccessType** (supplied) specifies the type of physical media access. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kSDLCAccess** indicates SDLC access. This is the default for this parameter.

♦ *Note:* At the time of publication, SDLC access was the only valid access type.

**noMonTimer** (supplied) specifies in terms of seconds the monitor interval for the MacAPPC Server. The valid range is 1 to 60. The default value for this parameter is 10. The **kIgnoreParam** constant indicates that the parameter is not specified.



**noNodeMsgs** (supplied) specifies whether or not the function of sending node operator messages is enabled. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that the node operator messages are not supported.

**kFuncSupp** indicates that the node operator messages are supported. This value is the default for this parameter.

- ❖ *Note:* Disabling node-operator message functions inhibits feed-back messages that report the completion of actions requested by way of node operator routines.

**noLogMsgs** (supplied) specifies whether or not the function of sending log messages is enabled. The following values are defined.

**kIgnoreParam** indicates that the parameter is not specified.

**kFuncNotSupp** indicates that log messages are not supported.

**kFuncSupp** indicates that log messages are supported. This value is the default for this parameter.

- ❖ *Note:* Disabling log message functions inhibits feedback messages that report error and informational status within the MacAPPC server.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also**      **NODisplayNode**



---

## NODefineStation

### Summary

The `NODefineStation` routine defines the station name and other parameters that pertain to the adjacent-link station (ALS).

This routine is used to initialize and modify operating parameters that are located in the new or existing station. The first time it is executed, this routine initializes the station definition with default values and updates it with the specified operating parameters. On subsequent executions this routine updates the station definition with the data supplied for the specified operating parameters. If a parameter is not specified, the value currently defined for that parameter remains unchanged. This routine can be executed by any transaction program that has the privilege to execute definition routines.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
0036	long	→	noCPName
002E	long	→	noALSName
003E	long	→	noPhoneNumber
0042	long	→	noALSAddr

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noLineName** (supplied) specifies a pointer to a string that contains the name of a previously defined line. The string length must not be greater than the value of the `kMaxName` constant. If the station has already been defined, this pointer can be `NIL` to indicate that the line name is to be left unchanged.

**noCPName** (supplied) specifies a pointer to a string that contains the name of a previously defined control point. The string length must not be greater than the value of the `kMaxName` constant. If the station has already been defined, this pointer can be `NIL` to indicate that the control point name is to be left unchanged.

**noALSName** (supplied) specifies a pointer to a string that contains the name of the adjacent link station (ALS) being defined or modified. The string length must not be greater than the value of the `kMaxName` constant. The string `LOCAL` is a reserved name designating intranode flows and must not be used as the name of an ALS.

**noPhoneNumber** (supplied) specifies a pointer to a string that contains the phone number to be displayed on the dial-out message. The string length must not be greater than the value of the `kMaxPhoneNumber` constant.



**noALSAddr** (supplied) specifies a pointer to a string that contains the station address. The string must be exactly as long as the value of the `kMaxSDLCAddr` constant, and the characters in the string must be hexadecimal. The parameter is used to specify the address of stations on a multipoint line. The default is C1. The values 00 and FF are invalid. If the value of this pointer is NIL, the current value is left unchanged.

<b>Result code</b>	<code>appcNoErr</code>	Routine succeeded
	<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
	<code>appcExec</code>	Routine executing; asynchronous request not complete

**See also** `NODisplayStation`



---

## NODElete

### Summary

The NODElete routine deletes the specified node component from a MacAPPC server. Defined components must be deleted in the reverse order in which they were defined; that is, stations must be deleted before lines and control points. Additionally, stations must not be deleted if they are active or have any modes defined that refer to them.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
0036	long	→	noCPName
002E	long	→	noALSName

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noLineName** (supplied) specifies a pointer to a string that contains the name of a line to be deleted. The string length must not be greater than the value of the `kMaxName` constant. This line must not be specified as the line parameter for a currently defined adjacent link station (ALS). The control point LOCAL cannot be deleted.

**noCPName** (supplied) specifies a pointer to a string that contains the name of the control point (CP) to be deleted. The string length must not be greater than the value of the `kMaxName` constant. This control point must not be specified as the control point for a currently defined ALS.

**noALSName** (supplied) specifies a pointer to a string that contains the name of an ALS to be deleted. The string length must not be greater than the value of the `kMaxName` constant. This ALS must not be specified as the ALS of a currently defined mode. The adjacent link station LOCAL cannot be deleted.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
appcExec	Routine executing; asynchronous request not complete



---

## NODisplayCP

### Summary

The NODisplayCP routine returns addressing information for a control point defined with NODefineCP. See NODefineCP for more information on the meaning of values returned here, particularly for specific information on the format of the noExchID and noCPUID parameters.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0062	byte	→	noNextCPName
0036	long	⇒	noCPName
0046	long	⇒	noExchID
004A	long	⇒	noCPUID

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noNextCPName** (supplied) specifies that information for the next control point should be returned. This parameter is used only if the noCPName parameter is specified. The following values are defined:

kIgnoreParam indicates that the parameter is not specified.

kNextEntry indicates that the information for the next control point should be returned.

**noCPName** (supplied/modified) specifies a pointer to space where the control point name can be returned. The space must be at least as long as the value of the kMaxName constant plus 1 byte. If the pointer points to a NULL string, information on the first control point, including its name, is returned. If a name is specified and the noNextCPName parameter is set to the value of the kNextEntry constant, information for the next control point, including its name, is returned. If the returned name is a NULL string, there are no more control points. If the noNextCPName parameter is set to kIgnoreParam, information for the specified control point is returned.

**noExchID** (supplied/modified) specifies a pointer to space where the defined XID can be returned. The space must be at least as long as the value of the kMaxExchID constant plus 1 byte. If the pointer is NIL, the XID is not returned. Since the XID is only defined for a peer, a NULL string is returned if the control point is a host.

**noCPUID** (supplied/modified) specifies a pointer to space where the defined CPU ID can be returned. The space must be at least as long as the value of the kMaxCPUID constant plus 1 byte. If the pointer is NIL, the CPU ID is not returned. Since the CPU ID is only defined for a host, a NULL string is returned if the control point is a peer.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

NODefineCP



---

## NODisplayLine

### Summary

The NODisplayLine routine returns information about a line defined with the NODefineLine routine. See NODefineLine for more information on the meaning of values returned here.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0061	byte	→	noNextLineName
0032	long	⇒	noLineName
0068	long	→	noLinePtr
0066	byte	←	noLineType
0065	byte	←	noLineStatus
----	byte	←	sdlcLineNum
----	byte	←	sdlcRoleType
----	byte	←	sdlcConnType
----	word	←	sdlcMaxBTU
----	word	←	sdlcLineSpeed
----	word	←	sdlcMaxRetry
----	word	←	sdlcIdleTime
----	word	←	sdlcNPRcvTime
----	word	←	sdlcMaxIFrame
----	byte	←	sdlcNRZIType
----	byte	←	sdlcDuplexType

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noNextLineName** (supplied) specifies that information for the next line should be returned. This parameter is used only if the **noLineName** parameter is supplied. The following values are defined:

**kIgnoreParam** indicates that the parameter is not specified.

**kNextEntry** indicates that the information for the next line should be returned.

**noLineName** (supplied/modified) specifies a pointer to space where the name of the line can be specified. The space must be at least as long as the value of the **kMaxName** constant plus 1 byte. If the pointer points to a NULL string, information for the first configured line, including the line name, is returned. If a name is specified, and the **noNextLineName** parameter is set to the value of the **kNextEntry** constant, the name of the next line is returned here. If the returned string is NULL, there are no more lines. If **noNextLineName** is set to the **kIgnoreParam** constant, information for the named line is returned.

**noLinePtr** (supplied) specifies a pointer to a line record (APPCLineRec).



**noLineType** (returned) specifies the type of line. The following values can be returned:

kSDLCLine indicates an SDLC line type.

❖ *Note:* At the time of publication, this is the only line type supported.

**noLineStatus** (returned) indicates the status of the line for which data is being returned. The following values can be returned:

kLineReset indicates the line status is reset.

kLinePendActive indicates the line status is pending active.

kLineActive indicates the line status is active.

kLinePendReset indicates the line status is pending reset.

**sdlcLineNum** (returned) indicates the SDLC line number. The following values can be returned:

kSDLCLine1 indicates SDLC line 1.

kSDLCLine2 indicates SDLC line 2.

kSDLCLine3 indicates SDLC line 3.

kSDLCLine4 indicates SDLC line 4.

**sdlcRoleType** (returned) indicates the SDLC role. The following values can be returned:

kSDLCPrimary indicates that the SDLC role is primary.

kSDLCSecondary indicates that the SDLC role is secondary.

kSDLCNegotiable indicates that the SDLC role is negotiable.

**sdlcConnType** (returned) indicates the SDLC connection type. The following values can be returned:

kSDLCLeased indicates that the connection is leased.

kSDLCMultiPoint indicates that the connection is multipoint.

kSDLCSwitched indicates that the connection is switched.

**sdlcMaxBTU** (returned) indicates the maximum basic transmission unit (BTU) length that can be received on this line.

**sdlcLineSpeed** (returned) indicates the SDLC line speed. The following values can be returned:

kSDLC300 indicates a line speed of 300 bps.

kSDLC1200 indicates a line speed of 1200 bps.

kSDLC2400 indicates a line speed of 2400 bps.

kSDLC4800 indicates a line speed of 4800 bps.

kSDLC9600 indicates a line speed of 9600 bps.

kSDLC19200 indicates a line speed of 19200 bps.

**sdlcMaxRetry** (returned) indicates the maximum number of times a frame is retransmitted after SDLC procedures have detected a discrepancy.



**sdlcIdleTime** (returned) indicates the amount of time (in milliseconds) that can elapse without a response before a primary link station initiates recovery action.

**sdlcNPRcvTime** (returned) indicates the amount of time (in milliseconds) that can elapse before the primary link station reports to a higher level of SNA that a nonproductive receive condition exists.

**sdlcMaxIFrame** (returned) is the maximum number of I-frames that can be sent before polling resumes.

**sdlcNRZIType** (returned) indicates which transmission encoding method is used when sending signals over this link. The following values can be returned.

kSDLCNRZ indicates that nonreturn-on-zero (NRZ) encoding is used.

kSDLCNRZI indicates that nonreturn-on-zero-inverted (NRZI) encoding is used.

**sdlcDuplexType** (returned) indicates whether data can be sent in one direction or both directions simultaneously without turning the line around. The following values can be returned.

kSDLCHalfDuplex indicates half-duplex transmissions. Data can be sent in only one direction at any given time.

kSDLCFullDuplex indicates full-duplex transmissions. Data can be sent and received without turning the line around.

<b>Result code</b>	appcNoErr	Routine succeeded
	appcFail	Routine failed; look in appcHiResult and appcLoResult
	appcExec	Routine executing; asynchronous request not complete

**See also** NODefineLine



---

## NODisplayNode

### Summary

The `NODisplayNode` routine returns information about an instance of the MacAPPC server. See `NODefineNode` for more information on the meaning of individual fields that are returned.

<b>Parameters</b>	000C	long	→	ioCompletion
	0018	word	→	appcRefNum
	001A	word	→	appcOpCode
	0022	long	→	appcUserRef
	0010	word	←	ioResult
	001C	word	←	appcHiResult
	001E	word	←	appcLoResult
	0026	long	→	noTPCBPtr
	0046	long	⇒	noExchID
	0058	byte	←	noAccessType
	0056	word	←	noMonTimer
	0063	byte	←	noNodeMsgs
	0064	byte	←	noLogMsgs

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noExchID** (supplied/modified) specifies a pointer to space where the defined XID can be returned. The space must be at least as long as the value of the `kMaxExchID` constant plus 1 byte. If the pointer is NIL, the XID is not returned.

**noAccessType** (returned) indicates the type of physical media access. The following values can be returned:

`kSDLCAccess` indicates SDLC access.

❖ *Note:* At the time of publication, SDLC access is the only valid access type.

**noMonTimer** (returned) indicates in seconds the monitor interval for the MacAPPC server.

**noNodeMsgs** (returned) indicates whether or not the function of sending node-operator messages is enabled. The following values can be returned:

`kFuncNotSupp` indicates that node-operator messages are not enabled.

`kFuncSupp` indicates that node-operator messages are enabled.

**noLogMsgs** (returned) indicates whether or not the function of sending log messages is enabled. The following values can be returned:

`kFuncNotSupp` indicates that log messages are not enabled.

`kFuncSupp` indicates that log messages are enabled.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`NODefineNode`



---

## NODisplayStation

### Summary

The `NODisplayStation` routine returns information about a station defined with `NODefineStation`. See `NODefineStation` for information on the meaning of values returned here.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	⇒	noLineName
0036	long	⇒	noCPName
0060	byte	→	noNextALSName
002E	long	⇒	noALSName
003E	long	⇒	noPhoneNumber
0042	long	⇒	noALSAddr
0067	byte	←	noALSStatus

### Description

**noTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**noLineName** (supplied/modified) specifies a pointer to space where the name of the line for this station can be returned. The space must be at least as long as the value of the `kMaxName` constant plus 1 byte. If the pointer is NIL, the line name is not returned.

**noCPName** (supplied/modified) specifies a pointer to space where the name of the control point for this station can be returned. The space must be at least as long as the value of the `kMaxName` constant plus 1 byte. If the pointer is NIL, the control point name is not returned.

**noNextALSName** (supplied) specifies that information for the next station should be returned. This parameter is used only if the `noALSName` parameter is specified. The following values are defined:

`kIgnoreParam` indicates that the parameter is not specified.

`kNextEntry` indicates that the information for the next station should be returned.

**noALSName** (supplied/modified) specifies a pointer to space where the name of the adjacent link station (ALS) can be specified. The space must be at least as long as the value of the `kMaxName` constant plus 1 byte. If the pointer points to a NULL string, information for the first configured station, including the station name, is returned. If a name is specified, and the `noNextALSName` parameter is set to `kNextEntry`, the name of the next station is returned. If the returned name is NULL, there are no more stations to display.

**noPhoneNumber** (supplied/modified) specifies a pointer to space where the phone number or dialing code defined for this station can be returned. The space must be at least as long as the value of the `kMaxPhoneNumber` constant plus 1 byte. If the pointer is NIL, the phone number is not returned.



**noALSAddr** (supplied/modified) specifies a pointer to space where the station address defined for this station can be returned. The space must be at least as long as the value of the **kMaxSDLCAddr** constant plus 1 byte.

**noALSStatus** (returned) indicates the status of the station for which data is returned. The following values can be returned:

**kStationReset** indicates that the station status is reset.

**kStationPendResp** indicates that the station status is active pending response.

**kStationPendCont** indicates that the station status is active pending contact.

**kStationActive** indicates that the station status is active.

**kStationPendReset** indicates that the station status is pending reset.

**kStationResetPendResp** indicates that the station status is reset pending response.

<b>Result code</b>	<b>appcNoErr</b>	Routine succeeded
	<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
	<b>appcExec</b>	Routine executing; asynchronous request not complete

**See also** **NODefineStation**



---

---

## Summary of the MacAPPC Node Operator Driver

This section provides a summary of the constants, data structures, and routines for use with the MacAPPC Node Operator Driver.

---

### Constants

The following constants are available for use with the MacAPPC Node Operator Driver.

```
{ noDialType values }

kConnectDial =          0;
kDialInOnDial =        1;
kDisconnectDial =       0;
kDialInOffDial =        1;

{ noQueueClass values }

kNoChangeQClass =       0;
kNodeOperMsgsQClass =   1;
kLogMsgsQClass =        2;
kDevelMsgsQClass =      4;
kTraceMsgsQClass =      8;

{ noQueueType }

kNoChangeQType =        0;
kInfoMsgsQType =        1;
kNotifMsgsQType =       2;
kErrorMsgsQType =       4;
kDiagMsgsQType =        8;

{ noQueueSevType values }

kNoChangeQSev =          0;
kReservedQSev =          1; { reserved }
kDevelMsgsQSev =        10;
kLowLevelInfoMsgsQSev = 20;
kNormalInfoMsgsQSev =   30;
kErrorMsgsQSev =         40;
kProgErrorsQSev =       90;

{ noAccessType values }

kSDLCAccess =           0;
```



```

{ noLineStatus values }

kLineReset =                1;
kLinePendActive =           2;
kLineActive =                3;
kLinePendReset =            4;

{ noLineType values }

kSDLCLine =                  0;

{ noStationStatus values }

kStationReset =              1;
kStationPendResp =           2;
kStationPendCont =           3;
kStationActive =             4;
kStationPendReset =          5;
kStationResetPendResp =      6;

{ sdlcLineNum values }

kSDLCLine1 =                 1;
kSDLCLine2 =                 2;
kSDLCLine3 =                 3;
kSDLCLine4 =                 4;

{ sdlcRoleType values }

kSDLCSecondary =             0;
kSDLCPrimary =               1;
kSDLCNegotiable =           2;

{ sdlcConnType values }

kSDLCLeased =                0;
kSDLCMultiPoint =            1;
kSDLCSwitched =              2;

{ sdlcDuplexType values }

kSDLCFullDuplex =            0;
kSDLCHalfDuplex =            1;

{ sdlcLineSpeed values }

kSDLC300 =                   300;
kSDLC1200 =                   1200;
kSDLC2400 =                   2400;
kSDLC4800 =                   4800;
kSDLC9600 =                   9600;
kSDLC19200 =                  19200;

{ sdlcNRZIType values }

kSDLCNRZ =                   0;
kSDLCNRZI =                   1;

```



---

## Data types

The following data types are available for use with the MacAPPC Node Operator Driver.

```
noParam:
(
noTPCBPtr      : Ptr;          { TPCB pointer }
noLclLUName    : StringPtr;    { local LU name pointer }
noALSName      : StringPtr;    { adjacent link station name pointer }
noLineName     : StringPtr;    { line name pointer }
noCPName       : StringPtr;    { control point name pointer }
noQueueName    : StringPtr;    { queue name pointer }
noPhoneNumber  : StringPtr;    { phone number pointer }
noALSAddr      : StringPtr;    { ALS Address pointer }
noExchID       : StringPtr;    { exchange ID pointer }
noCPUID        : StringPtr;    { CPU ID pointer }
noCorrID       : INTEGER;      { correlation ID }
noDataPtr      : Ptr;          { data buffer pointer }
noDataSize     : INTEGER;      { data buffer length }
noMonTimer     : INTEGER;      { wakeup timer for monitor }
noAccessType   : SignedByte;   { Access Type }
noStopSrvr     : SignedByte;   { halt server }
noDialType     : SignedByte;   { dial type }
noQueueEnable  : SignedByte;   { queue enabled }
noQueueClass   : SignedByte;   { queue class }
noQueueType    : SignedByte;   { queue type }
noQueueSev     : SignedByte;   { queue severity level }
noWaitForMsg   : SignedByte;   { block if no message in queue }
noNextALSName  : SignedByte;   { next adjacent link station name }
noNextLineName : SignedByte;   { next line name }
noNextCPName   : SignedByte;   { next CP name }
noNodeMsgs     : SignedByte;   { enable node messages }
noLogMsgs      : SignedByte;   { enable logging messages }
noLineStatus   : SignedByte;   { line status }
noLineType     : SignedByte;   { line type }
noALSStatus    : SignedByte;   { station status }
noLinePtr      : Ptr;          { line structure pointer }
);
```

### APPC line record

```
APPCLineType =      (sdlcLine);
```

```
APPCLineRec =      RECORD
CASE
sdlcLine:
(
sdlcMaxBTU      :  INTEGER;      { maximum BTU length }
```



```

sdlcMaxRetry      :  INTEGER;      { maximum retries }
sdlcIdleTime      :  INTEGER;      { idle time before recovery }
sdlcNPRcvTime     :  INTEGER;      { non-productive receive time }
sdlcMaxIFrame     :  INTEGER;      { maximum I-frames before polling }
sdlcLineSpeed     :  INTEGER;      { line speed }
sdlcLineNum       :  SignedByte;   { Line Number }
sdlcRoleType      :  SignedByte;   { SDLC role }
sdlcConnType      :  SignedByte;   { connection type }
sdlcDuplexType    :  SignedByte;   { duplex type }
sdlcNRZIType      :  SignedByte;   { NRZI support }
);

```

```
END;
```

```
APPCLineRecPtr = ^APPCLineRec;
```



---

## Node control routines

The following node control routines are available for use with the MacAPPC Node Operator Driver.

### NOActivateLine

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
004E	long	←	noCorrID

### NOActivateLU

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
002A	long	→	noLclLUName
004E	long	←	noCorrID

### NOActivateNode

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
004E	long	←	noCorrID

### NOActivateStation

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
002E	long	→	noALSName
005A	byte	→	noDialType
004E	long	←	noCorrID



**NODEactivateLine**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
004E	long	←	noCorrID

**NODEactivateLU**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
002A	long	→	noLclLUName
004E	long	←	noCorrID

**NODEactivateNode**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0059	byte	→	noStopSrvr
004E	long	←	noCorrID

**NODEactivateStation**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
002E	long	→	noALSName
005A	byte	→	noDialType
004E	long	←	noCorrID



---

## Node message routines

The following node message routines are available for use with the MacAPPC Node Operator Driver.

### NODefineMessageQueue

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
003A	long	→	noQueueName
005B	byte	→	noQueueEnable
005C	byte	→	noQueueClass
005D	byte	→	noQueueType
005E	byte	→	noQueueSev

### NODisplayMessage

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
003A	long	→	noQueueName
0050	long	⇒	noDataPtr
0054	word	→	noDataSize
005F	byte	→	noWaitForMsg
004E	long	→	noCorrID

### NODisplayMessageQueue

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
003A	long	→	noQueueName
005B	byte	←	noQueueEnable
005C	byte	←	noQueueClass
005D	byte	←	noQueueType
005E	byte	←	noQueueSev



---

## Node definition routines

The following node definition routines are available for use with the MacAPPC Node Operator Driver.

### NODefineCP

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0036	long	→	noCPName
0046	long	→	noExchID
004A	long	→	noCPUID

### NODefineLine

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
0068	long	→	noLinePtr
0066	byte	→	noLineType
----	byte	→	sdlcLineNum
----	byte	→	sdlcRoleType
----	byte	→	sdlcConnType
----	word	→	sdlcMaxBTU
----	word	→	sdlcLineSpeed
----	word	→	sdlcMaxRetry
----	word	→	sdlcIdleTime
----	word	→	sdlcNPRcvTime
----	word	→	sdlcMaxIFrame
----	byte	→	sdlcNRZIType
----	byte	→	sdlcDuplexType

### NODefineNode

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr



0046	long	→	noExchID
0058	byte	→	noAccessType
0056	word	→	noMonTimer
0063	byte	→	noNodeMsgs
0064	byte	→	noLogMsgs

#### **NODefineStation**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
0036	long	→	noCPName
002E	long	→	noALSName
003E	long	→	noPhoneNumber
0042	long	→	noALSAddr

#### **NODelete**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	→	noLineName
0036	long	→	noCPName
002E	long	→	noALSName

#### **NODisplayCP**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0062	byte	→	noNextCPName
0036	long	⇒	noCPName
0046	long	⇒	noExchID
004A	long	⇒	noCPUID

#### **NODisplayLine**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult



001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0061	byte	→	noNextLineName
0032	long	⇒	noLineName
0068	long	→	noLinePtr
0066	byte	←	noLineType
0065	byte	←	noLineStatus
----	byte	←	sdlcLineNum
----	byte	←	sdlcRoleType
----	byte	←	sdlcConnType
----	word	←	sdlcMaxBTU
----	word	←	sdlcLineSpeed
----	word	←	sdlcMaxRetry
----	word	←	sdlcIdleTime
----	word	←	sdlcNPRcvTime
----	word	←	sdlcMaxIFrame
----	byte	←	sdlcNRZIType
----	byte	←	sdlcDuplexType

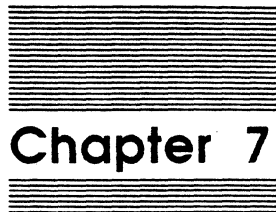
#### **NODisplayNode**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0046	long	⇒	noExchID
0058	byte	←	noAccessType
0056	word	←	noMonTimer
0063	byte	←	noNodeMsgs
0064	byte	←	noLogMsgs

#### **NODisplayStation**

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0026	long	→	noTPCBPtr
0032	long	⇒	noLineName
0036	long	⇒	noCPName
0060	byte	→	noNextALSName
002E	long	⇒	noALSName
003E	long	⇒	noPhoneNumber
0042	long	⇒	noALSAddr
0067	byte	←	noALSStatus





## **Chapter 7**

# **MacAPPC Transaction Program Driver**



This chapter describes the MacAPPC Transaction Program Driver (TP62), explains how to use the driver, and provides a detailed guide to the programmatic interface for executing each Transaction Program Driver routine. For quick reference, a section at the end of the chapter summarizes the data structures, constants, and routine parameters.

---

## Using the MacAPPC Transaction Program Driver

The transaction program routines are used to establish a connection between a transaction program and the MacAPPC server, as well as to convert strings from the ASCII character set to EBCDIC and vice versa.

---

### Getting the currently selected MacAPPC server

When a transaction program executes a `TPAttach` routine, the `tpSrvrEntiryPtr` parameter must point to an AppleTalk entity name structure identifying a particular MacAPPC server. The MacAPPC Chooser device maintains the currently selected MacAPPC server as an entity name resource inside of the Chooser device's own resource fork. The following Pascal procedure shows a method of filling in an entity name structure with the information about the currently selected MacAPPC server.

```

PROCEDURE GetChosenSrvr(SrvrEntPtr : EntityPtr);
{ Get currently chosen MacAPPC server }
{ It's in resource type SENT, ID 500 in the MacAPPC Chooser file }

VAR
    fRefNum:      INTEGER;      { refNum of MacAPPC Chooser file }
    oldRefNum:    INTEGER;      { refNum of current resource file }
    entityHndl:   Handle;       { handle to entity resource }
    entityPtr:    EntityPtr;    { pointer to entity resource }
    theWorld:     SysEnvRec;     { for SysEnvirons }

BEGIN
    { call SysEnvirons to get path to System Folder }
    rc := SysEnvirons(1, theWorld);
    IF (rc <> noErr) THEN
    BEGIN
        ShowError('Error calling SysEnvirons!');
        Exit(PROGRAM);
    END;

    { save current resource file }
    oldRefNum := CurResFile;

    { open MacAPPC Chooser file 'MacAPPC' }
    fRefNum := OpenRFPPerm('MacAPPC', theWorld.sysVRefNum, fsRdPerm);
    IF (fRefNum = -1) THEN
    BEGIN
        ShowError('Could not open MacAPPC file!');
        Exit(PROGRAM);
    END;

    { get the server entity resource }
    entityHndl := GetResource('SENT', 500);
    IF (entityHndl = NIL) THEN
    BEGIN
        ShowError('Could not get currently chosen server!');
        CloseResFile(fRefNum);
        Exit(PROGRAM);
    END;

```



```

END;

HLock(entityHndl);
entityPtr := POINTER(ORD4(entityHndl^));

{ copy it to specified entity }
SrvrEntPtr^.objStr := entityPtr^.objStr;
SrvrEntPtr^.typeStr := entityPtr^.typeStr;
SrvrEntPtr^.zoneStr := entityPtr^.zoneStr;

{ Although it's not done here, you may want to check to      }
{ make sure the objStr is not null, i.e. there is no currently }
{ chosen server.  You can then warn the user of this instead of }
{ having the attach fail later.}

{ get rid of resource and close file }
HUnlock(entityHndl);
ReleaseResource(entityHndl);
CloseResFile(fRefNum);
UseResFile(oldRefNum);

END; { GetChosenSrvr }

```

## Attaching and its implications

When you choose one of the various kinds of attaches, you essentially determine what kind of MacAPPC routines you will be able to execute for that attach. The routines that can be executed for the various attaches are shown more precisely in Table 7-1.

**Table 7-1**  
Routines valid for different attach types

Routines	Server	LU	Wait
Mapped conversation		X	X
Type-independent conversation		X	X
Basic conversation		X	X
Change-number-of-sessions		X	X
Session control		X	X
LU definition	X	X	X
Node control	X		
Node message	X		
Node definition	X		
TP connection	Attach not required		
TP utility	Attach not required		



---

---

## MacAPPC transaction program routines

This section describes the MacAPPC transaction program routines. The routines are divided into the following categories:

- connection routines, which are used to establish and terminate connections to MacAPPC servers
- utility routines, which are used for general transaction program utility functions

---

---

## Transaction program connection routines

This section describes the MacAPPC transaction program connection routines. These routines connect a transaction program to a particular MacAPPC server or a particular LU within the MacAPPC server, or wait for an allocation request specifying a particular transaction program and connect to that conversation.



---

## TPAttach

### Summary

The `TPAttach` routine initiates communication between the local program and the MacAPPC server.

---

### Important

The `TPAttach` routine must be the first MacAPPC routine executed by the local transaction program.

If the local transaction program is waiting for a remote transaction program to start the conversation, the local transaction program must execute a `TPAttach` routine with the `tpAttachType` parameter set to the `kWaitAttach` constant.

If the local transaction program is starting a conversation, it must first execute a `TPAttach` routine with the `tpAttachType` parameter set to the `kLUAttach` constant and then execute a `MCAAllocate` or `BCAllocate` routine (depending on the type of conversation) before it executes any other conversation routine. See `MCAAllocate` or `BCAllocate` in Chapter 4 of this manual.

---

The type of attach, as specified in the `tpAttachType` parameter, determines the type of MacAPPC routines that can be used during the attach. See the section "Attaching and Its Implications," earlier in this chapter, for more information.

### Parameters

000C	long	→	<code>ioCompletion</code>
0018	word	→	<code>appcRefNum</code>
001A	word	→	<code>appcOpCode</code>
0022	long	→	<code>appcUserRef</code>
0010	word	"	<code>ioResult</code>
001C	word	"	<code>appcHiResult</code>
001E	word	"	<code>appcLoResult</code>
0020	byte	←	<code>appcConvState</code>
0026	long	→	<code>tpTPCBPtr</code>
002A	long	→	<code>tpCVCBPtr</code>
0034	long	→	<code>tpPIPBufPtr</code>
0038	word	→	<code>tpPIPBufSize</code>
002E	long	→	<code>tpMapBufPtr</code>
0032	word	→	<code>tpMapBufSize</code>
0050	byte	→	<code>tpAttachType</code>
003E	long	→	<code>tpLclProgName</code>
003A	long	→	<code>tpLclLUName</code>
0042	long	→	<code>tpSrvrEntityPtr</code>
004E	word	→	<code>tpWaitTime</code>
0052	long	→	<code>tpMapProc</code>
005C	----	⇒	<code>tpPIPPtr[]</code>
045C	----	↔	<code>tpPIPSize[]</code>
0046	long	←	<code>tpConvID</code>
004A	long	←	<code>tpProgID</code>

### Description

**`tpTPCBPtr`** (supplied) specifies a pointer to a Transaction Program Control Block (TPCB) whose length is determined by the value of the `kTPCBSIZE` constant. You must supply a new TPCB each time you execute a `TPAttach` routine.



**tpCVCBPtr** (supplied) specifies a pointer to a Conversation Control Block (CVCB) whose length is determined by the value of the **kCVCBSize** constant. You must supply a new CVCB each time you execute a **TPAttach** routine when the **tpAttachType** parameter is set to the **kWaitAttach** constant.

**tpPIPBuffPtr** (supplied) specifies a pointer to a buffer that holds the program-initialization parameters. The length of the buffer is specified by the value of the **tpPIPBuffSize** parameter. This parameter is required only when the **tpAttachType** parameter is set to the **kWaitAttach** constant and the conversation will start with PIP data.

**tpPIPBuffSize** (supplied) specifies the size of the buffer pointed to by **tpPIPBuffPtr**. This buffer must be large enough to hold the largest amount of PIP data expected plus a 4-byte logical length ID (LLID) per parameter plus one 4-byte LLID for the entire PIP data.

**tpMapBuffPtr** (supplied) specifies a pointer to a mapped conversation buffer. The length of the buffer is specified by the value of the **tpMapBuffSize** parameter. This parameter is required only when the **tpAttachType** parameter is set to the **kWaitAttach** constant and the conversation will be a mapped conversation.

**tpMapBuffSize** (supplied) specifies the size of the mapped conversation buffer pointed to by the **tpMapBuffPtr** parameter. This buffer must be large enough to hold the largest complete data record expected plus a 4-byte LLID.

**tpAttachType** (supplied) specifies the type of attach. The following values are defined:

**kSrvrAttach** indicates that an attach is being made to a MacAPPC server.

**kLUAttach** indicates that an attach is being made to an LU.

**kWaitAttach** indicates that an allocation is being initiated by another TP, and that this TP should wait for allocation.

**tpLclProgName** (supplied) specifies a pointer to a string that contains the name of the local transaction program. The string length must not be greater than the value of the **kMaxTPName** constant. This parameter is required only when the **tpAttachType** parameter is set to the **kLUAttach** or **kWaitAttach** constant.

**tpLclLUName** (supplied) specifies a pointer to a string that contains the name of the local LU that the local transaction program will use. The string length must not be greater than the value of the **kMaxName** constant. This parameter is required only when the **tpAttachType** parameter is set to the **kLUAttach** or **kWaitAttach** constant.

**tpSrvrEntityPtr** (supplied) specifies a pointer to an AppleTalk entity name structure that contains information about the name, type, and zone of a MacAPPC server.

**tpWaitTime** (supplied) specifies the maximum time in seconds that the MacAPPC Server should wait for a conversation routine to complete before timing out the routine. The following additional values are defined:

**kMaxWait** indicates no maximum wait time.

**kConfigWait** indicates that the default value configured for the local LU is to be used.



**tpMapProc** (supplied) specifies a pointer to a mapping procedure; the mapping procedure is executed whenever the conversation requires it. This value applies only when the **tpAttachType** parameter is set to the **kWaitAttach** and the conversation will be a mapped conversation. Set the parameter to **NIL** to use the default mapping procedure.

**tpPIPPtr** (supplied/modified) specifies space for an array of pointers to program-initialization parameters. The maximum number of parameters is defined by the value of the **kMaxPIP** constant, with a total space limitation equal to the value of the **tpPIPBuffSize** parameter. This value applies only when the **tpAttachType** parameter is set to the **kWaitAttach** constant.

**tpPIPSize** (supplied/returned) specifies an array of sizes that specifies the size for each PIP in the **tpPIPPtr** parameter. This value applies only when the **tpAttachType** parameter is set to the **kWaitAttach** constant. The actual size of each PIP is returned in the **tpPIPPtr** parameter. If the value of **tpPIPSize** is smaller than the value for the largest PIP returned, the data is truncated. The last size is followed by a size of 0. This value applies only when the **tpAttachType** parameter is set to the **kWaitAttach** constant.

**tpConvID** (returned) indicates the conversation ID. This value applies only when the **tpAttachType** parameter is set to the **kWaitAttach** constant.

**tpProgID** (returned) indicates the transaction program ID for the local transaction program.

#### Notes

On successful return from a **TPAttach** routine when the **tpAttachType** parameter is set to the **kWaitAttach** constant, the conversation is in receive state.

#### Result code

<b>appcNoErr</b>	Routine succeeded
<b>appcFail</b>	Routine failed; look in <b>appcHiResult</b> and <b>appcLoResult</b>
<b>appcExec</b>	Routine executing; asynchronous request not complete

#### See also

**TPDetach**



---

## TPDetach

### Summary

The `TPDetach` routine detaches the current connection from the MacAPPC server.

Any conversations still open under a detached connection are deallocated by the MacAPPC server.

---

### Important

Your transaction program must execute a `TPDetach` routine to terminate an existing connection. After the `TPDetach` routine has been executed, no more routines can be executed for that detached connection.

---

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	tpTPCBPtr
0051	byte	→	tpDetachType

### Description

**tpTPCBPtr** (supplied) specifies a pointer to an existing Transaction Program Control Block (TPCB).

**tpDetachType** (supplied) specifies the type of detach. The following values are defined:

`kNormalDetach` specifies that the connection to the MacAPPC server is to be terminated normally. The MacAPPC server will be notified of the detach and will be able to take normal clean-up procedures.

`kAbortDetach` specifies that the connection to the MacAPPC server should be terminated immediately without notifying the MacAPPC server. Use this detach type to terminate the connection when an asynchronous MacAPPC routine is outstanding.

### Result code

<code>appcNoErr</code>	Routine succeeded
<code>appcFail</code>	Routine failed; look in <code>appcHiResult</code> and <code>appcLoResult</code>
<code>appcExec</code>	Routine executing; asynchronous request not complete

### See also

`TPAttach`



---

---

## Transaction program utility routines

This section describes the MacAPPC transaction program utility routines. These routines are used to perform utility functions, such as the conversion of ASCII strings to EBCDIC and from EBCDIC to ASCII.



---

## TPAsciiToEbcdic

### Summary

The TPAsciiToEbcdic routine translates an ASCII string into EBCDIC (the IBM character set). A transaction program can use it to translate application-specific information for use by partner programs that run on IBM machines.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0056	long	⇒	tpDataPtr
005A	word	→	tpDataSize

### Description

**tpDataPtr** (supplied/modified) is a pointer to the start of the data to be translated.

**tpDataSize** (supplied) specifies the number of bytes to be translated.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

TPEbcdicToAscii



---

## TPEbcdicToAscii

### Summary

The TPEbcdicToAscii routine translates a string from EBCDIC (the IBM character set) into ASCII. A transaction program can use it to translate application-specific information sent by partner programs that run on IBM machines.

### Parameters

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0056	long	⇒	tpDataPtr
005A	word	→	tpDataSize

### Description

**tpDataPtr** (supplied/modified) is a pointer to the start of the data to be translated.

**tpDataSize** (supplied) specifies the number of bytes to be translated.

### Result code

appcNoErr	Routine succeeded
appcFail	Routine failed; look in appcHiResult and appcLoResult
appcExec	Routine executing; asynchronous request not complete

### See also

TPAsciiToEbcdic



---

---

## Summary of the MacAPPC Transaction Program Driver

This section provides a summary of the constants, data structures, and routines for use with the MacAPPC Transaction Program Driver.

---

### Constants

The following constants are available for use with the MacAPPC Transaction Program Driver.

```
{ tpAttachType values }
```

```
kLUAttach =          0;
```

```
kSrvrAttach =        1;
```

```
kWaitAttach =        2;
```

```
{ tpDetachType values }
```

```
kNormalDetach =      0;
```

```
kAbortDetach =       1;
```

```
{ tpWaitTime values }
```

```
kMaxWait =           -1;      { wait time = forever }
```

```
kConfigWait =        0;      { wait time = local LU wait time }
```



---

## Data types

The following data types are available for use with the MacAPPC Transaction Program Driver.

```
tpParam:
(
tpTCBPTr      : Ptr;          { TPCB pointer }
tpCVCBPTr     : Ptr;          { CVCB pointer }
tpPIPBuffPtr  : Ptr;          { PIP buffer pointer }
tpPIPBuffSize : INTEGER;      { PIP buffer size }
tpMapBuffPtr  : Ptr;          { mapped conversation buffer pointer }
tpMapBuffSize : INTEGER;      { mapped conversation buffer size }
tpLclLUName   : StringPtr;    { local LU name pointer }
tpLclProgName : StringPtr;    { local program name pointer }
tpSrvrEntityPtr : EntityPtr;  { server entity pointer }
tpConvID      : LONGINT;      { conversation ID }
tpProgID      : LONGINT;      { transaction program ID }
tpWaitTime    : INTEGER;      { wait time in seconds }
tpAttachType  : SignedByte;   { attach type }
tpDetachType  : SignedByte;   { detach type }
tpMapProc     : ProcPtr;      { mapping procedure pointer }
tpDataPtr     : Ptr;          { data buffer pointer }
tpDataSize    : INTEGER;      { data buffer size }
tpPIPPtr      : ARRAY[1..kMaxPIP] OF Ptr; { array of PIP ptrs }
tpPIPSize     : ARRAY[1..kMaxPIP] OF INTEGER; { array of PIP sizes }
);
```



---

## Connection routines

The following connection routines are available for use with the MacAPPC Transaction Program Driver.

### TPAttach

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	tpTPCBPtr
002A	long	→	tpCVCBPtr
0034	long	→	tpPIPBuffPtr
0038	word	→	tpPIPBuffSize
002E	long	→	tpMapBuffPtr
0032	word	→	tpMapBuffSize
0050	byte	→	tpAttachType
003E	long	→	tpLclProgName
003A	long	→	tpLclLUName
0042	long	→	tpSrvrEntityPtr
004E	word	→	tpWaitTime
0052	long	→	tpMapProc
005C	----	⇒	tpPIPPtr[]
045C	----	↔	tpPIPSize[]
0046	long	↔	tpConvID
004A	long	←	tpProgID

### TPDetach

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0020	byte	←	appcConvState
0026	long	→	tpTPCBPtr
0051	byte	→	tpDetachType



---

## Utility routines

The following utility routines are available for use with the MacAPPC Transaction Program Driver.

### TPAsciiToEbcDic

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0056	long	⇒	tpDataPtr
005A	word	→	tpDataSize

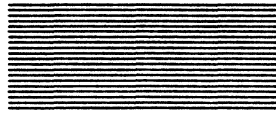
### TPEbcDicToAscii

000C	long	→	ioCompletion
0018	word	→	appcRefNum
001A	word	→	appcOpCode
0022	long	→	appcUserRef
0010	word	←	ioResult
001C	word	←	appcHiResult
001E	word	←	appcLoResult
0056	long	⇒	tpDataPtr
005A	word	→	tpDataSize

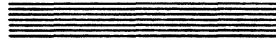








## **Chapter 8**



### **MacAPPC Example TP**



This chapter presents Pascal code that shows an example fragment of a MacAPPC transaction program. This fragment illustrates what a typical set of MacAPPC routines look like. For a complete example of a working transaction program, see the program SampleTP.p on your distribution disk.

- ❖ *Note:* Because of limitations for the line length of this guide, some of the error and status messages in the following code have been artificially wrapped to the next line without a continuation symbol.

```
{-----}

FILE
    ExampleTP.p
    Copyright Apple Computer, Inc. 1988
    All rights reserved.

NAME
    ExampleTP

DESCRIPTION
    A small example MacAPPC transaction program demonstrating simple
    mapped conversations. It can be used with the LocalConfig
    provided with MacAPPC. After starting the server, make sure you activate
    the local LUs and user modes. Run this example TP on two client machines.

    As an example, here are some parameters that will work with LocalConfig:

    ExampleTP1                ExampleTP2
    -----
    Local LU = LU3            Local LU = LU1
    Local TP = TP3            Remote LU = LU2
    Click Wait button         Mode = MODE1
                                Remote TP = TP3
                                Local TP = TP1
                                Message = Any string to send to ExampleTP1
                                Click Allocate button *after* ExampleTP1 is
waiting
                                waiting

    From a MacAPPC standpoint, the action looks like:

    ExampleTP1                ExampleTP2
    -----
    Wait attach
                                LU attach
                                MCAAllocate
                                MCSendData
                                MCDeallocate(flush)
                                Detach LU attach
    MReceiveAndWait for message
    MReceiveAndWait to get deallocation indication
    MCDeallocate(local)
    Detach Conv attach

    -----}
{$R-} { Turn off range checking - you must always do this for MacAPPC }
```

```
PROGRAM ExampleTP;
```

```
    USES
```

```
        MemTypes, Quickdraw, OSIntf, ToolIntf, PackIntf,
        AppleTalk, APPC, APPCErrors;
```

```
    CONST
```

```
        TPDialog          = 128;    { ID of dialog }
        QuitButton        = 1;      { items in dialog }
        AllocateButton    = 2;
        WaitButton        = 3;
        LocalLUText       = 13;
        RemoteLUText      = 14;
```



```

ModeText          = 15;
RemoteTPText      = 16;
LocalTPText       = 17;
MessageText       = 18;
StatusText        = 21;
OutlineItem       = 22;
HLineItem         = 23;
ErrorALRT         = 129;  { id of error alert }

```

```

VAR
theDialog:      DialogPtr; { dialog box }
itemHit:        INTEGER; { returned from ModalDialog }
itemType:       INTEGER; { used to manipulate dialog items }
item:           Handle; { used to manipulate dialog items }
box:            Rect; { used to manipulate dialog items }
LocalLUName:    Str255; { local LU for Allocate }
RemoteLUName:   Str255; { remote LU for Allocate }
ModeName:       Str255; { mode for Allocate }
RemoteTPName:   Str255; { remote TP for Allocate }
LocalTPName:    Str255; { local TP for Allocate and Wait }
MessageStr:     Str255; { message for Allocate }
MajorStr:       Str255; { string representation of major code }
MinorStr:       Str255; { string representation of minor code }
StatusStr:      Str255; { string for current status }
rc:             OSErr; { generic OSErr }
TPBPBPtr:      APPCParamBlockPtr; { global TPPB }
CVPBPBPtr:     APPCParamBlockPtr; { global CVPB }
TPCBPtr:       Ptr; { global TPCB - need one more for Wait }
CVCBPtr:       Ptr; { global CVCB }
MapBuffPtr:    Ptr; { global map buffer for mapped conversations }
SrvrEntPtr:    EntityPtr; { currently chosen MacAPPC server }
result:        Boolean; { dummy boolean }

```

```

PROCEDURE ShowError(errorStr : Str255);
{ Put up error alert with specified string }

```

```

VAR
    itemh: INTEGER;

BEGIN
    ParamText(errorStr, '', '', '');
    itemh := StopAlert(ErrorALRT, NIL);

```

```

END; { ShowError }

```

```

PROCEDURE ShowStatus(statusStr : Str255);
{ Set status text in dialog }

```

```

BEGIN
    GetDItem(theDialog, StatusText, itemType, item, box);
    SetIText(item, statusStr);

```

```

END; { ShowStatus }

```

```

PROCEDURE InitMacAPPC;
{ Allocate parameter blocks and other memory, open drivers }

```

```

VAR
    TPRefNum: INTEGER; { refNum of .TP62 driver }
    CVRefNum: INTEGER; { refNum of .CV62 driver }

```

```

BEGIN
    { open TP and CV drivers to get refnums }
    rc := OpenDriver('.TP62', TPRefNum);
    IF (rc <> noErr) THEN
    BEGIN
        ShowError('Could not open .TP62 driver!');
        Exit(PROGRAM);
    END;

```



```

rc := OpenDriver('.CV62', CVRefNum);
IF (rc <> noErr) THEN
BEGIN
    ShowError('Could not open .CV62 driver!');
    Exit(PROGRAM);
END;

{ allocate TP and CV parameter blocks }
TPBPPtr := APPCParamBlockPtr(NewPtr(kTPSize));
CVBPPtr := APPCParamBlockPtr(NewPtr(kCVSize));

{ allocate other necessary memory }
SrvrEntPtr := EntityPtr(NewPtr(SIZEOF(EntityName)));
TPCBPtr := NewPtr(kTPCBSIZE);
CVCBPtr := NewPtr(kCVCBSIZE);

{ allocate map buffer for mapped conversation }
MapBuffPtr := NewPtr(kMapBuffSize);

{ fill in the parameter blocks with refnums }
{ this will be the only field we don't fill in every time }
TPBPPtr^.appcRefNum := TPRefNum;
CVBPPtr^.appcRefNum := CVRefNum;

END; { InitMacAPPC }

FUNCTION DoLUAttach(LocalLUName, LocalTPName : Str255) : Boolean;
{ Attach to a local LU in preparation for an allocate }
{ Returns TRUE if successful, FALSE otherwise }

BEGIN

    DoLUAttach := TRUE;
    ShowStatus('Doing LU attach...');

    { fill in parameter block }
    TPBPPtr^.appcOpCode := kTPAttach;
    TPBPPtr^.tpTPCBPtr := TPCBPtr;
    TPBPPtr^.tpAttachType := kLUAttach;
    TPBPPtr^.tpLclProgName := @LocalTPName;
    TPBPPtr^.tpLclLUName := @LocalLUName;
    TPBPPtr^.tpSrvrEntityPtr := SrvrEntPtr;
    TPBPPtr^.tpWaitTime := kMaxWait;

    rc := PBControl(ParmBlkPtr(TPBPPtr), FALSE);

    IF (rc = noErr) THEN
    BEGIN
        ShowStatus('LU attach successful.');
```

```

    END
    ELSE
    BEGIN
        NumToString(LONGINT(TPBPPtr^.appcHiResult), MajorStr);
        NumToString(LONGINT(TPBPPtr^.appcLoResult), MinorStr);
        StatusStr := Concat('Error doing LU attach, major = ',
                             MajorStr, ', minor = ', MinorStr, '.');
        ShowError(StatusStr);
        DoLUAttach := FALSE;
    END;

END; { DoLUAttach }

FUNCTION DoWaitAttach(LocalLUName, LocalTPName : Str255) : Boolean;
{ Do a wait attach in preparation }
{ to be allocated. TP will block on wait until allocated. }
{ Returns TRUE if successful, FALSE otherwise }
```



```

BEGIN

    DoWaitAttach := TRUE;

    { do wait attach }
    ShowStatus('Doing wait attach...');

    { fill in parameter block }
    TPPBPtr^.appcOpCode := kTPAttach;
    TPPBPtr^.tpPCBPtr := TPCBPtr;
    TPPBPtr^.tpCVCBPtr := CVCBPtr;
    TPPBPtr^.tpPIPBuffPtr := NIL;
    TPPBPtr^.tpPIPBuffSize := 0;
    TPPBPtr^.tpAttachType := kWaitAttach;
    TPPBPtr^.tpLclLUName := @LocalLUName;
    TPPBPtr^.tpLclProgName := @LocalTPName;
    TPPBPtr^.tpSrvrEntityPtr := SrvrEntPtr;
    TPPBPtr^.tpWaitTime := kMaxWait;
    TPPBPtr^.tpPIPPtr[1] := NIL;
    TPPBPtr^.tpPIPSize[1] := 0;
    TPPBPtr^.tpMapProc := NIL;

    rc := PBControl(ParmBlkPtr(TPPBPtr), FALSE);

    IF (rc = noErr) THEN
        ShowStatus('Wait attach successful.')
    ELSE
        BEGIN
            NumToString(LONGINT(TPPBPtr^.appcHiResult), MajorStr);
            NumToString(LONGINT(TPPBPtr^.appcLoResult), MinorStr);
            StatusStr := Concat('Error doing wait attach, major = ',
                               MajorStr, ', minor = ', MinorStr, '.');
            ShowError(StatusStr);
            DisposPtr(waitTPCBPtr);
            DoWaitAttach := FALSE;
            Exit(DoWaitAttach);
        END;
    END; { DoWaitAttach }

FUNCTION DoDetach : Boolean;
{ Detach current conv or LU attach }
{ Returns TRUE if successful, FALSE otherwise }

BEGIN

    DoDetach := TRUE;
    ShowStatus('Doing detach...');

    { fill in parameter block }
    TPPBPtr^.appcOpCode := kTPDetach;
    TPPBPtr^.tpPCBPtr := TPCBPtr;
    TPPBPtr^.tpDetachType := kNormalDetach;

    rc := PBControl(ParmBlkPtr(TPPBPtr), FALSE);

    IF (rc = noErr) THEN
        ShowStatus('Detach successful.')
    ELSE
        BEGIN
            NumToString(LONGINT(TPPBPtr^.appcHiResult), MajorStr);
            NumToString(LONGINT(TPPBPtr^.appcLoResult), MinorStr);
            StatusStr := Concat('Error doing detach, major = ',
                               MajorStr, ', minor = ', MinorStr, '.');
            ShowError(StatusStr);

            DoDetach := FALSE;
        END;
    END; { DoDetach }

FUNCTION DoMCAAllocate(RemoteLUName, ModeName, TPName : Str255) : Boolean;

```



```

{ Allocate a mapped conversation to the specified TP at the }
{ specified remote LU over the specified mode }
{ Returns TRUE if successful, FALSE otherwise }

BEGIN
    DoMCAAllocate := TRUE;
    ShowStatus('Doing MCAAllocate...');

    { fill in parameter block }
    CVPBPTr^.appcOpCode := kMCAAllocate;
    CVPBPTr^.cvTPCBPtr := TPCBPtr;
    CVPBPTr^.cvCVCBPtr := CVCBPtr;
    CVPBPTr^.cvMapBuffPtr := MapBuffPtr;
    CVPBPTr^.cvMapBuffSize := kMapBuffSize;
    CVPBPTr^.cvPIPBuffPtr := NIL;
    CVPBPTr^.cvPIPBuffSize := 0;
    CVPBPTr^.cvRmtLUName := @RemoteLUName;
    CVPBPTr^.cvRmtProgName := @TPName;
    CVPBPTr^.cvModeName := @ModeName;
    CVPBPTr^.cvUserName := NIL;
    CVPBPTr^.cvUserPswd := NIL;
    CVPBPTr^.cvUserProf := NIL;
    CVPBPTr^.cvReturnCtl := kWhenAllocReturn;
    CVPBPTr^.cvSyncType := kNoSync;
    CVPBPTr^.cvPIPUsed := FALSE;
    CVPBPTr^.cvPIPPtr[1] := NIL;
    CVPBPTr^.cvPIPSize[1] := 0;
    CVPBPTr^.cvSecType := kNoSec;
    CVPBPTr^.cvMapProc := NIL;

    rc := PBControl(ParmBlkPtr(CVPBPTr), FALSE);

    IF (rc = noErr) THEN
        BEGIN
            ShowStatus('MCAAllocate successful.');
        END
    ELSE
        BEGIN
            NumToString(LONGINT(CVPBPTr^.appcHiResult), MajorStr);
            NumToString(LONGINT(CVPBPTr^.appcLoResult), MinorStr);
            StatusStr := Concat('Error doing MCAAllocate, major = ',
                               MajorStr, ', minor = ', MinorStr, '.');
            ShowError(StatusStr);
            DoMCAAllocate := FALSE;
        END;
    END; { DoMCAAllocate }

FUNCTION DoMCSendData(MessageStr : Str255) : Boolean;
{ Send specified message over current conversation }
{ Returns TRUE if successful, FALSE otherwise }

VAR
    buffer:      Str255;      { buffer for send data }
    mapname:     Str255;      { map name }

BEGIN
    DoMCSendData := TRUE;

    { set up buffer }
    buffer := MessageStr;

    ShowStatus('Doing MCSendData...');

    { fill in parameter block }
    CVPBPTr^.appcOpCode := kMCSendData;
    CVPBPTr^.cvTPCBPtr := TPCBPtr;
    CVPBPTr^.cvCVCBPtr := CVCBPtr;
    CVPBPTr^.cvDataPtr := Ptr(@buffer);
    CVPBPTr^.cvDataSize := Length(MessageStr) + 1;
    { this TP does not use map names }

```



```

mapname := '';
CVPBPtr^.cvMapName := StringPtr(@mapname);
CVPBPtr^.cvFMHdrs := FALSE;

rc := PBControl(ParmBlkPtr(CVPBPtr), FALSE);

IF (rc = noErr) THEN
    ShowStatus('MCSendData successful.')
ELSE
    BEGIN
        NumToString(LONGINT(CVPBPtr^.appcHiResult), MajorStr);
        NumToString(LONGINT(CVPBPtr^.appcLoResult), MinorStr);
        StatusStr := Concat('Error doing MCSendData, major = ',
                             MajorStr, ', minor = ', MinorStr, '.');
        ShowError(StatusStr);
        DoMCSendData := FALSE;
    END;
END; { DoMCSendData }

FUNCTION DoMCReceiveAndWait(VAR MessageStr : Str255) : Boolean;
{ Wait for a message over current conversation }
{ Does a second receive to get deallocate indication }
{ Returns TRUE if successful, FALSE otherwise }

VAR
    buffer:      Str255;      { buffer for received data }
    mapname:     Str255;      { map name }

BEGIN
    DoMCReceiveAndWait := TRUE;

    { initialize buffer }
    buffer := '';

    ShowStatus('Doing MCReceiveAndWait for data...');

    { fill in parameter block }
    CVPBPtr^.appcOpCode := kMCReceiveAndWait;
    CVPBPtr^.cvTPCBPtr := TPCBPtr;
    CVPBPtr^.cvCVCBPtr := CVCBPtr;
    CVPBPtr^.cvDataPtr := Ptr(@buffer);
    CVPBPtr^.cvDataSize := 256;
    { shouldn't be sent a map name, but we should provide a buffer for it }
    CVPBPtr^.cvMapName := StringPtr(@mapname);

    rc := PBControl(ParmBlkPtr(CVPBPtr), FALSE);

    IF ((rc = noErr) AND (CVPBPtr^.cvWhatRcvd = kDataComplRcvd)) THEN
        BEGIN
            ShowStatus('MCReceiveAndWait for data successful.');
            MessageStr := buffer;
        END
    ELSE
        BEGIN
            IF (rc <> noErr) THEN
                BEGIN
                    NumToString(LONGINT(CVPBPtr^.appcHiResult), MajorStr);
                    NumToString(LONGINT(CVPBPtr^.appcLoResult), MinorStr);
                    StatusStr := Concat('Error doing MCReceiveAndWait for
                                         data, major = ', MajorStr, ', minor = ', MinorStr, '.');
                END
            ELSE
                BEGIN
                    { got unexpected what received }
                    NumToString(LONGINT(CVPBPtr^.cvWhatRcvd), MajorStr);
                    StatusStr := Concat('Error doing MCReceiveAndWait for
                                         data, cvWhatRcvd = ', MajorStr, '.');
                END;
            ShowError(StatusStr);
            DoMCReceiveAndWait := FALSE;
            Exit(DoMCReceiveAndWait);
        END
    END
END;

```



```

END;

{got the data, now we must do one more receive to see the deallocation }
ShowStatus('Doing MCRceiveAndWait for deallocate...');

{ fill in parameter block }
CVPBPTr^.appcOpCode := kMCRceiveAndWait;
CVPBPTr^.cvTPCBPtr := TPCBPtr;
CVPBPTr^.cvCVCBPtr := CVCBPtr;
CVPBPTr^.cvDataPtr := Ptr(@buffer);
CVPBPTr^.cvDataSize := 256;
CVPBPTr^.cvMapName := StringPtr(@mapname);

rc := PBControl(ParmBlkPtr(CVPBPTr), FALSE);

IF ((rc <> noErr) AND (CVPBPTr^.appcHiResult = deallocErr) AND
    (CVPBPTr^.appcLoResult = normDeallocErr)) THEN
BEGIN
    ShowStatus('MCRceiveAndWait for deallocate successful.');
```

```

END
ELSE
BEGIN
    IF (rc <> noErr) THEN
    BEGIN
        NumToString(LONGINT(CVPBPTr^.appcHiResult), MajorStr);
        NumToString(LONGINT(CVPBPTr^.appcLoResult), MinorStr);
        StatusStr := Concat('Error doing MCRceiveAndWait for
                             deallocate, major = ',
                             MajorStr, ', minor = ', MinorStr, '.');
```

```

    END
    ELSE
    BEGIN
        { got unexpected no error }
        StatusStr := 'Error doing MCRceiveAndWait for deallocate,
                     got no error.';

    END;
    ShowError(StatusStr);
    DoMCRceiveAndWait := FALSE;
END;

END; { DoMCRceiveAndWait }

FUNCTION DoMCDeallocate(deallocType : SignedByte) : Boolean;
{ deallocate current conversation with the specified type }
{ Returns TRUE if successful, FALSE otherwise }

BEGIN
    DoMCDeallocate := TRUE;
    ShowStatus('Doing MCDeallocate...');

    { fill in parameter block }
    CVPBPTr^.appcOpCode := kMCDeallocate;
    CVPBPTr^.cvTPCBPtr := TPCBPtr;
    CVPBPTr^.cvCVCBPtr := CVCBPtr;
    CVPBPTr^.cvDeallocType := deallocType;

    rc := PBControl(ParmBlkPtr(CVPBPTr), FALSE);

    IF (rc = noErr) THEN
        ShowStatus('MCDeallocate successful.')
```

```

    ELSE
    BEGIN
        NumToString(LONGINT(CVPBPTr^.appcHiResult), MajorStr);
        NumToString(LONGINT(CVPBPTr^.appcLoResult), MinorStr);
        StatusStr := Concat('Error doing MCDeallocate, major = ',
                             MajorStr, ', minor = ', MinorStr, '.');
```

```

        ShowError(StatusStr);
        DoMCDeallocate := FALSE;
    END;
END; { DoMCDeallocate }

```



```

BEGIN                                { main program }

{ initialize everything }
InitGraf(@thePort);
InitFonts;
FlushEvents(everyEvent, 0);
InitWindows;
TEInit;
InitDialogs(NIL);
InitCursor;

{ open drivers and allocate needed memory }
InitMacAPPC;

{ get currently chosen server from MacAPPC Chooser device }
{ the listing for this routine is in a different Chapter }
GetChosenSrvr(SrvrEntPtr);

theDialog := GetNewDialog(TPDialog, NIL, POINTER(-1));

{ set up status }
ShowStatus('Please fill in parameters and choose a function. ');
ShowWindow(WindowPtr(theDialog));

WHILE TRUE DO
BEGIN
    itemHit := 0;

    CASE itemHit OF
        QuitButton:
        BEGIN
            { just exit }
            DisposDialog(theDialog);
            Exit(PROGRAM);
        END;

        AllocateButton:
        BEGIN
            { allocating a conv - need both LUs, mode, and both TPs }
            { also need message to send }
            GetDItem(theDialog, RemoteLUText, itemType, item, box);
            GetIText(item, RemoteLUName);
            GetDItem(theDialog, LocalLUText, itemType, item, box);
            GetIText(item, LocalLUName);
            GetDItem(theDialog, ModeText, itemType, item, box);
            GetIText(item, ModeName);
            GetDItem(theDialog, RemoteTPText, itemType, item, box);
            GetIText(item, RemoteTPName);
            GetDItem(theDialog, LocalTPText, itemType, item, box);
            GetIText(item, LocalTPName);
            GetDItem(theDialog, MessageText, itemType, item, box);
            GetIText(item, MessageStr);

            { attach to LU first }
            IF (DoLUAttach(LocalLUName, LocalTPName)) THEN
            BEGIN
                { allocate conversation }
                IF (DoMCAAllocate(RemoteLUName, ModeName,
                                   RemoteTPName)) THEN
                BEGIN
                    { send message }
                    result := DoMCSendData(MessageStr);

                    { deallocate, flushing buffer }
                    result := DoMCDeallocate(kFlushDealloc);
                END;

                { finally, detach }
                result := DoDetach;
            END; { IF DoLUAttach }
        END; { AllocateButton }
    END;
END;

```



```

WaitButton:
BEGIN
    { waiting to be allocated - need local LU name and }
    { local TP name }
    GetDItem(theDialog, LocalTPText, itemType, item, box);
    GetIText(item, LocalTPName);
    GetDItem(theDialog, LocalLUText, itemType, item, box);
    GetIText(item, LocalLUName);

    { wait for allocation }
    IF (DoWaitAttach(LocalLUName, LocalTPName)) THEN
    BEGIN
        { receive data }
        IF (DoMCReceiveAndWait(MessageStr)) THEN
        BEGIN
            { got message, put in dialog }
            GetDItem(theDialog, MessageText,
                    itemType, item, box);
            SetIText(item, MessageStr);
        END;

        { deallocate conversation }
        result := DoMCDeallocate(kLocalDealloc);

        { detach }
        result := DoDetach;
    END; { IF DoWaitAttach }
    END; { WaitButton }
    END; { CASE itemHit OF }
    END; { WHILE TRUE }

END. { program }

```





## **Part III**

# **MacAPPC User's Guide**

This part consists of four chapters that provide step-by-step instructions on how to setup and run a MacAPPC network. These chapters assume little or no knowledge of MacAPPC beyond that provided in Part I, "Introduction."

Chapter 9 describes how to set up the hardware and software necessary for a MacAPPC installation. Hardware installation sections are provided for the client computer, the server computer, and the communications card. Software installation sections are provided for the client computer and the server computer, as well as the MacAPPC Configuration and Administration programs.

Chapter 10 describes the use of the Chooser desk accessory to select MacAPPC servers.

Chapter 11 describes the Configuration program and how it is used to create a configuration file that defines the components of a MacAPPC network.

Chapter 12 describes the Administration program and how it is used to monitor and control MacAPPC servers.

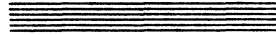








## **Chapter 9**



# **Installation**



This chapter provides instructions on how to set up the hardware and software necessary for a MacAPPC installation.

---

---

## Hardware

MacAPPC is based on a hardware platform that consists of an intelligent communications card that hosts a particular MacAPPC server, the server computer that contains the communications card, and client computers that are connected to the server computer on an AppleTalk network. Since the server computer can also act as a client computer, AppleTalk connections are not required.

---

### Client computer

The client computer can be a Macintosh Plus, Macintosh SE, or Macintosh II. The client computers can be connected to a server computer using any type of AppleTalk connection.

---

### Server computer

The server computer must be a Macintosh II. The server does not need to be dedicated to the MacAPPC server function and can also be used as a client, if desired.

---

### Communications card

The intelligent communications card is a NuBus™ card that contains a CPU, RAM, and one or more serial RS-232 ports. This card can be plugged into any of the available NuBus slots in the server computer. Refer to the Macintosh II owner's guide for a detailed explanation of how to install NuBus cards in your Macintosh II computer.

At the time of publication of this guide, the only commercially available card with the required specifications is the AST-ICP card manufactured by AST Research Inc. The AST-ICP card provides a Motorola 68000 CPU clocked at 7.3728 MHz, 512K of RAM with no wait states, and four ports configurable for four RS-232 serial connections. For serial communications, a four-port cable (AST part #220130-002) is provided.

Before installing an AST-ICP card in the Macintosh II, you must verify that the jumpers are properly set for serial communications. MacAPPC requires the jumpers to be set in the configuration described in Table 9-1. Refer to the *AST-ICP User's Manual* for detailed instructions on how to set the jumpers.



**Table 9-1**  
AST-ICP communications card jumper settings

Jumper	Factory setting	MacAPPC
E1	Left or Right	No change required
E2	Up	Down
E3	Up	Down
E4	On	On
E5	Right	Right
E6	On	On
E7	Right	Right
E8	On	On
E9	On	On
E10	Right	Right
E11	Left	Right
E12	Right	Right
E13	Left	Left
E14	Left	Right
E15	Left	Right

## Software

MacAPPC is based on a software platform consisting of a MacAPPC server process that operates under the Macintosh Coprocessor Platform™ (MCP™) operating system on an intelligent communications card. The MacAPPC server is able to communicate with remote processes via the AppleTalk Data Stream Protocol (ADSP) and the MacAPPC drivers. Selection of a particular MacAPPC server is accomplished using the MacAPPC Chooser device. See Chapter 10 for additional information on the MacAPPC Chooser. The MacAPPC Administration and Configuration programs provide a means for definition and administration of a MacAPPC server and its services.

The MacAPPC user's files are provided on the *MacAPPC User* disk and *MacAPPC System* disk, as summarized in Tables 9-2 and 9-3, respectively. All computers should use System version 6.0.2 and Finder version 6.1 or later versions.

**Table 9-2**  
MacAPPC User disk

File	Description
MacAPPC	MacAPPC Chooser device
APPCDrivers	MacAPPC device drivers
APPCServer	MacAPPC server
Admin	MacAPPC Administration program
Config	MacAPPC Configuration program
Local Config	An example of a local configuration file
Remote Config	An example of a remote configuration file
Admin Log	MacAPPC Administration program log file



**Table 9-3**  
MacAPPC System disk

File	Description
ADSP	AppleTalk ADSP device driver (for client computers)
Apple IPC	MCP operating system IPC device driver (for server computers)
FWD	MCP operating system ADSP forwarder (for server computers)

ADSP is a symmetric, connection-oriented protocol that makes possible the establishment and maintenance of full-duplex streams of data bytes between two sockets in an AppleTalk internet. Data flow on an ADSP connection is reliable and flow-controlled; ADSP guarantees ordering and delivery.

The MCP operating system is intended to be a base for the development of communications services on intelligent communications cards. It provides these services independent of the Macintosh computer hosting one or more of these cards. The Apple IPC (Inter-process Communications) component provides inter-process communication services between tasks running under the MCP operating system and programs running on the host Macintosh. The FWD (Forwarder) component provides a client-server interface between services running under the MCP operating system and clients connected to the server computer via AppleTalk ADSP connections.

---

## Client computer

The following files must be copied to the System Folder of the client computer:

- ☐ ADSP
- ☐ APPCDrivers
- ☐ MacAPPC

Insert the distribution disks, drag these icons to the System Folder, and restart the client computer. See the sections that follow on the Configuration and Administration programs.

---

## Server computer

The following files must be transferred to the System Folder of the server computer:

- ☐ Apple IPC
- ☐ FWD
- ☐ APPCServer
- ☐ ADSP
- ☐ APPCDrivers
- ☐ MacAPPC

Insert the distribution disks, drag these icons to the System Folder, and restart the server computer. See the sections that follow on the Configuration and Administration programs.



---

## **Configuration program**

The MacAPPC Configuration program (Config) may be used on any Macintosh computer. This program does not require the services of MacAPPC for the creation of configuration files. Make a copy of the Configuration program before you use it. Double-click the Config icon to start the application. See Chapter 11 for additional information.

---

## **Administration program**

The MacAPPC Administration program (Admin) must be used on the server computer and may be used on any client computer. Make a copy of the Administration program before you use it. Double-click the Admin icon to start the application. The MacAPPC Administration program must be executed from the server computer when starting or stopping a MacAPPC server, but it can be executed from any client for display and control of a MacAPPC server. See Chapter 12 for additional information.

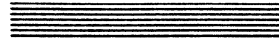








## **Chapter 10**



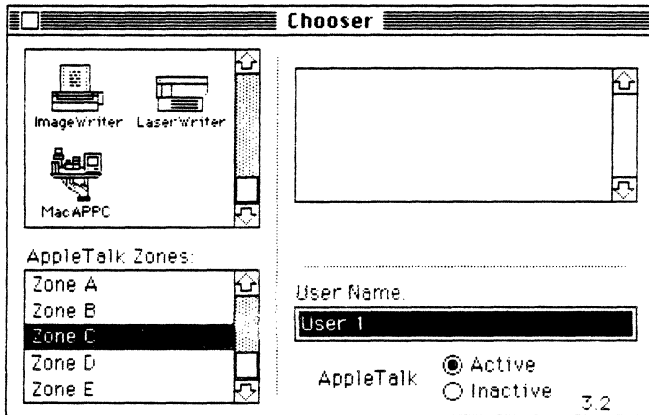
# **Selecting a MacAPPC Server**



This chapter explains how to select a MacAPPC server from the list of available servers on your AppleTalk network.

Selection of a MacAPPC server is made using the Chooser desk accessory under the Apple menu. MacAPPC extends the use of the Chooser to allow selection of a particular MacAPPC server from the list of registered servers on your AppleTalk network. The server may be either in the Macintosh that you are using (intranode) or in a Macintosh you are connected to via the AppleTalk network (internode).

To select a MacAPPC server, choose Chooser from the Apple menu. A chooser window similar to Figure 10-1 appears.

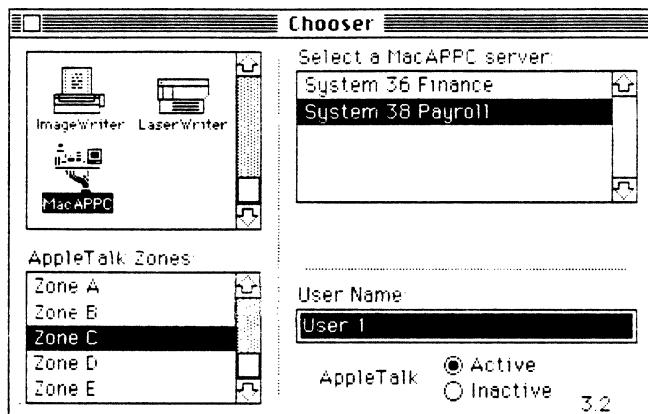


**Figure 10-1**  
The Chooser

The contents of the Chooser window varies, depending on the type of Chooser files you have installed in your System Folder. This example shows icons that allow selection of a number of printer devices and MacAPPC servers. If your network is divided into AppleTalk zones, the names of the zones appear in a separate list. If your AppleTalk network is not divided into zones, this list does not appear and the list of icons extends to the bottom of the Chooser window. MacAPPC requires that AppleTalk be active.



To display the list of MacAPPC servers on your AppleTalk network, select the MacAPPC icon and AppleTalk zone, if appropriate. A list of available servers appears under the heading "Select a MacAPPC server:" (see Figure 10-2). Select the server you want to use. This selection will remain in effect until you select another server.



**Figure 10-2**  
Selecting a MacAPPC server

For additional information on the Chooser and AppleTalk, see your Macintosh owner's guide.









## **Chapter 11**



# **The MacAPPC Configuration Program**



This chapter describes the Configuration program and how it is used to create a configuration file that defines the components of a MacAPPC network.

You must use the Configuration program before the Administration program to create a configuration file describing your MacAPPC network components. The Configuration program provides a clear and organized Macintosh user interface to assist in the network configuration and modification process.

The Configuration program allows you to create components with a predefined set of default values that reduces the number of options you must select for each component. Before creating components, you can edit these default values to minimize the number of individual changes you need to make later. Once components are created, you can edit them to complete the specification of their functional relationships, roles, and requirements in the network.

After the configuration file is created, you can use the Administration program to start a server and use the file to create an actual network configuration.

Network components are the elements that make up an SNA node. The Configuration program consists of facilities that permit you to define, delete, edit, and print out information on the following seven components:

- ☐ Node: This is the local node. There is one per file, that is, one per configuration.
- ☐ Local LU: This component holds information on a local LU, including the name by which it is known to other LUs and TPs across the network, and any conversation-level security (user IDs, passwords, and profiles) it may support.
- ☐ Transaction program: This component holds information on a transaction program (TP), including the local LU with which it is associated and any conversation-level security (user IDs, profiles, or both) it may support.
- ☐ Line: This component describes a physical link to other nodes.
- ☐ Partner: This component describes a partner node. *Partner* is a MacAPPC term that consists of both a station and a control point. It is used by the Configuration program to simplify configuration design and implementation.
- ☐ Remote LU: This component describes a remote, or partner, LU, whether it is in the same server or at a partner node. It is associated with a specific local LU and partner.
- ☐ Mode: This component describes a mode between a specific local LU-remote LU pair.

Many of these components are interdependent. For example, you must create a local LU before you can create a TP, because every TP must be associated with a specific local LU.

Each of these components is made up of settings that you must define in order to configure your system. There is one network configuration per file. To define a configuration, therefore, you must create a new configuration file or open an existing configuration file.

The following sections detail each component window and explain the settings that you must define. You first create network components, and then edit their respective settings to complete the network configuration.

To help you during the configuration process, Appendix K provides remote and local configuration worksheets.



---

---

## The Configuration program menu bar

The Configuration program has three menus in addition to the Apple menu: File, Edit, and Create. With the File menu, illustrated in Figure 11-1, you can create a new configuration file, open a previously created file, get information about a component, close the file, save the file, save the file under a different name, use the components and configurations that were created and edited in the last saved file, format a page to be printed, print the configuration file, or quit the configuration program.

The Edit menu, illustrated in Figure 11-2, has the standard Macintosh user interface commands for editing, plus a command for editing default settings. When you choose Edit Defaults, a submenu appears from which you can choose Node, Local LU, TP, Line, Partner, Remote LU, or Mode for default settings editing. When you choose a command from the submenu, the appropriate dialog box with the current defaults appears. You may change any or all of the component default settings. See the section on editing defaults, later in this chapter, for more details.

File	
New...	⌘N
Open...	⌘O
Get Info	⌘I
Close	⌘W
Save	⌘S
Save as...	
Revert to Saved	
Page Setup...	
Print...	
Quit	⌘Q

**Figure 11-1**  
The File menu

Edit	
Undo	⌘Z
Cut	⌘H
Copy	⌘C
Paste	⌘V
Clear	⌘B
Edit Defaults	▶
Node...	
Local LU...	
TP...	
Line...	
Partner...	
Remote LU...	
Mode...	

**Figure 11-2**  
The Edit menu



The Create menu, illustrated in Figure 11-3, is used to create network components and security options. Note that there is no command to create a local node. Since each file corresponds to a single configuration, creating a file implies creating a node.

The Configuration program allows you to choose only the menu items that are appropriate to the current stage of network configuration. Menu items that are dimmed cannot yet be used to create a component—you must first create or select the component on which it is dependent.

Create	
Local LU	⌘L
TP	⌘T
Line	
Partner	
Remote LU	⌘R
Mode	⌘M
<hr/>	
User	⌘U
Profile	⌘P

**Figure 11-3**  
The Create menu

---

---

## Conventions used in the Configuration program

This section describes some conventions used in the Configuration program that you should know about before proceeding.

---

### Screen and key conventions

Radio buttons are small circles that appear next to options in the windows and dialog boxes. To select an option, click the radio button next to the text or the text itself of the option you want. Only one radio button at a time can be selected for a group of options.

Check boxes are small boxes that appear next to options in windows and dialog boxes. To select an option, click the check box or the text of the option you want. A cross mark appears on the check box to indicate that it has been selected. More than one check box at a time can be selected for a group of options. To turn off a check box, click the check box. The cross mark will disappear.

The Tab key advances the insertion point to the next option.

The Return or Enter keys may be used as alternatives to clicking OK.



---

## Character type conventions

Symbol-strings are used to name component settings. Type A is a symbol-string type consisting of one or more uppercase letters *A* through *Z*; numerics 0 through 9; and special characters \$, #, and @; the first character of which is an uppercase letter or a special character. The following names all use type A characters: local LU name, line name, partner name, remote LU name, mode name, CNOS adjacent link station (ALS) name, local LU network name, remote LU network qualifier, local LU network qualifier, and remote LU network name.

Type AE is a symbol-string type consisting of one or more lowercase letters *a* through *z*; uppercase letters *A* through *Z*; numerics 0 through 9; special characters \$, #, @, and the period (.); with no restrictions on the first character. The following names all use type AE characters: user ID, password, remote LU password, profile, and transaction program name.

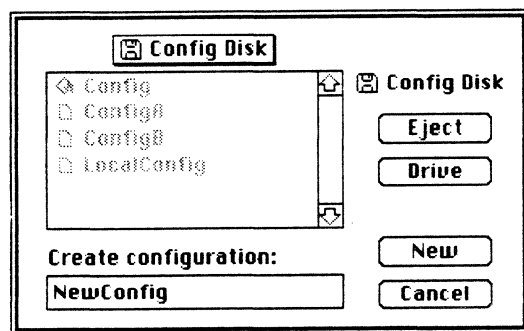
Type hexadecimal (hex) is a character type consisting of numbers in the base-16 system, using the ten digits 0 through 9 and the six letters A through F. The following fields all use hex characters: node exchange ID, remote LU password, transaction program network name, ALS address, and partner CPU ID or exchange ID.

---

---

## Creating a configuration file

To create a new configuration file, choose New from the File menu. A dialog box similar to the one in Figure 11-4 appears. Notice that a list of names of previously created configuration files stored on the active disk appears, but the names are dimmed. You cannot select any of the files in this list; they are displayed to remind you of the names that you have already used. (If you had chosen Open from the File menu instead of New, you would select one of the configuration files displayed in the list.)



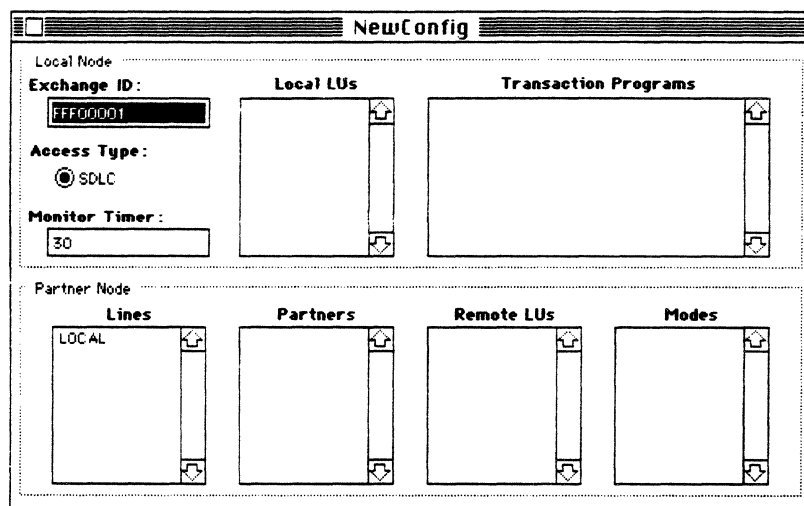
**Figure 11-4**  
Creating a new configuration file

You must enter a new configuration file name before proceeding. Configuration file names have up to 64 characters. The dialog box shows a new configuration file named *NewConfig*. Click New to begin the creation of a new MacAPPC configuration file. The configuration file window in Figure 11-5 appears.



Figure 11-5 shows a new configuration file window named *NewConfig*. Notice that there are lists for six of the seven network components: local LUs, transaction programs, lines, partners, remote LUs, and modes. Each configuration can have only one local node; the node information is shown in the file window. You create an instance of each component by using the Create menu; once a component has been created, its name appears in the appropriate list.

The configuration file window is divided into two sections, Local Node and Partner Node.



**Figure 11-5**  
The configuration file window

## Local node section

This section contains local node information and lists for local LUs and transaction programs.

### Exchange ID

Exchange ID contains the identifier of the local node if the node is a peer. The ID of the remote node is stored in the partner. It is used in XID exchange at link establishment. Exchange IDs have exactly eight hex (0-F) characters representing a 4-byte binary number. In IBM implementations, the first three characters are set based on the product type (03A for the Displaywriter, 03E for the System/36, and so forth).

### Access Type

Access Type specifies the type of line being defined. Synchronous Data Link Control (SDLC) is the only access type currently available.

### Monitor Timer

Monitor Timer specifies the wake-up interval for the program monitor in seconds. The wake-up interval is the time the server waits for a reply from an attached TP. If the interval is exceeded, the server disconnects. The valid range is 1 to 60. The default setting is 30.



---

## Partner node section

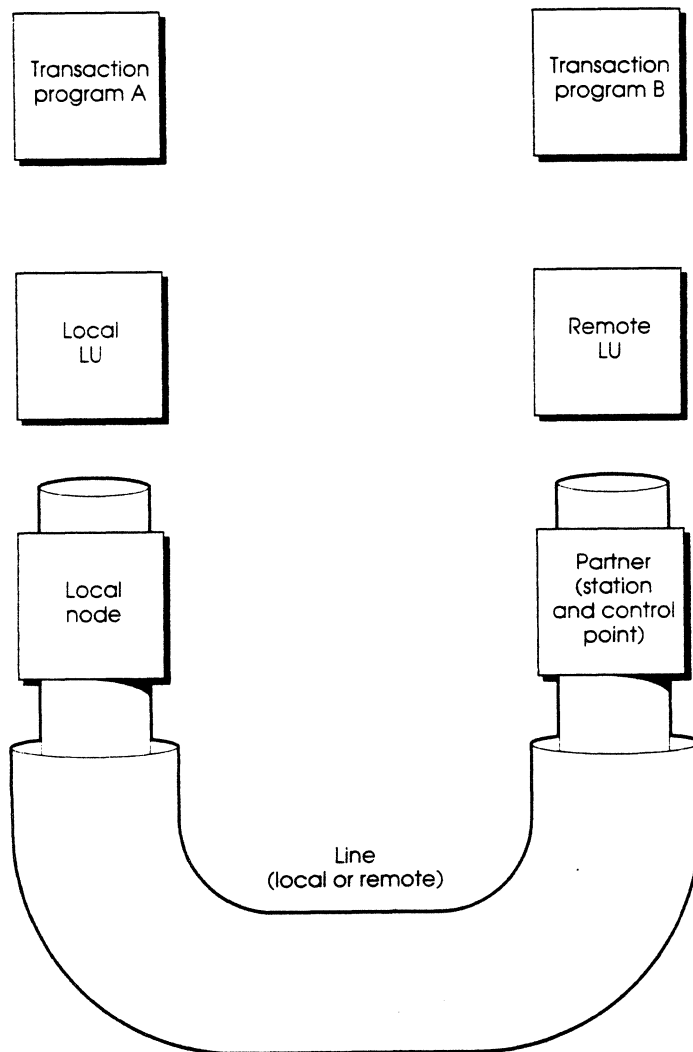
This section contains lists for lines, partners, remote LUs, and modes. The reserved line named *LOCAL* and its associated partner named *LOCAL* (not shown) are a part of every configuration file. Note that these are not actual lines or partners, but are metaphors for creating remote LUs and modes that exist within the local node. In other words, there is no physical link to another node; the network is entirely within the MacAPPC server.

---

---

## Creating network components

Figure 11-6 shows you a schematic outline of the network components of a configuration, to help you visualize the resources that need to be created and their relationship to each other.



**Figure 11-6**  
Configuration network components

As you progress through this section, this diagram is repeated to show the components that have been created and the relationships that are established.



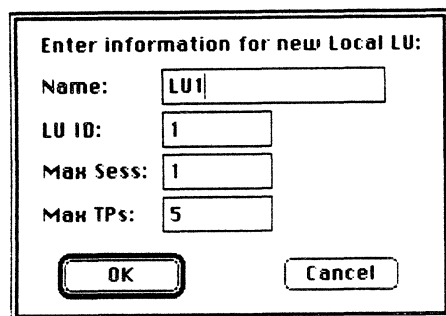
To create a network component, pull down the Create menu and choose the component you wish to create. The program displays a dialog box that prompts for the name of the component, along with the settings that describe it. Other settings are filled in from default values. As you create and edit components, their names appear in the lists in the file window. You must enter legal values in each field before the component can be created.

The Configuration program's error checking routine assists in finding logic and formatting errors before you enter the incorrect information. Prompts indicate problems you must correct before you can proceed. This error-checking feature is provided throughout the Configuration program.

---

## Creating local LUs

Local LUs can be created at any time. One or more local LUs in the local node must be created before TPs, remote LUs, or modes can be created. To create a local LU, choose Local LU from the Create menu. The dialog box in Figure 11-7 appears.



Enter information for new Local LU:

Name:

LU ID:

Max Sess:

Max TPs:

**Figure 11-7**  
Creating a local LU

### Name (Local LU name)

The local LU name is used to associate mode, remote LU, and transaction program definitions with the local LU. Local LU names have up to eight symbol-string type A characters. This example uses *LU1* for the local LU name.

### LU ID

LU ID is the numeric identifier for the local LU. Each local LU must have a unique LU ID. The value corresponds to the destination address field (DAF). The valid range is 1 to 254. The default setting is 1.

### Max Sess (Maximum number of sessions)

Max Sess specifies the maximum number of sessions that can be active at the local LU at one time. Session limits are currently maintained only at a mode level. The valid range is 0 to 254. Entering 0 indicates that this setting should be ignored. The default setting is 1.



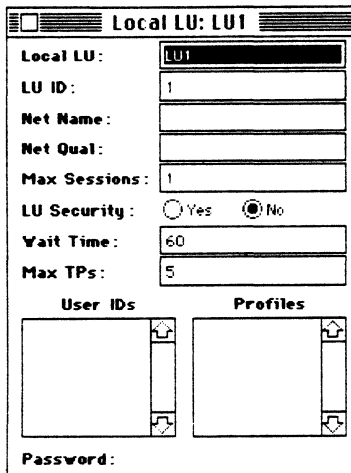
## Max TPs (Maximum number of TPs)

Max TPs specifies the maximum number of transaction programs that can be attached to the local LU at one time. The valid range is 1 to 255. The default setting is 5.

Click OK to create a new local LU. The new LU named *LU1* is added to the configuration file window. You may now create user IDs for the local LU.

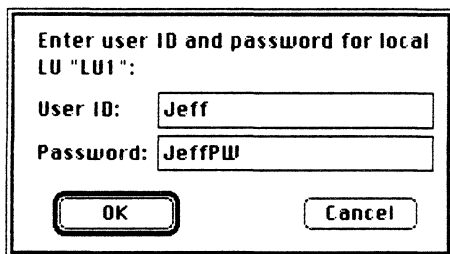
## User ID and password

To create a new user ID and password for a local LU, double-click to select a local LU from the configuration file window (see Figure 11-5). Alternatively, you may select the local LU and choose Get Info from the File menu. The window in Figure 11-8 appears.



**Figure 11-8**  
The local LU window

Choose User from the Create menu. Note that all of the commands except User are dimmed. A profile cannot be created until both a user ID and a password are specified. The dialog box in Figure 11-9 appears. This example shows that a User ID named *Jeff* with password *JeffPW* is being created for local LU *LU1*.



**Figure 11-9**  
Creating a new user ID and password for a local LU

**User ID:** A network security setting. A user ID has up to ten symbol-string type AE characters.

**Password:** A network security setting. A password has up to ten symbol-string type AE characters.



After creating a new user ID and password, the user ID appears in the local LU User IDs list. When you select a user ID on the list, the corresponding password is displayed at the bottom of the window, as shown in Figure 11-10. You may now create one or more profiles for the local LU user.

The image shows a window titled "Local LU: LU1". It contains several fields: "Local LU:" with a dropdown menu showing "LU1"; "LU ID:" with a text field containing "1"; "Net Name:" with an empty text field; "Net Qual:" with an empty text field; "Max Sessions:" with a text field containing "1"; "LU Security:" with two radio buttons, "Yes" and "No", where "No" is selected; "Wait Time:" with a text field containing "60"; and "Max TPs:" with a text field containing "5". Below these fields are two lists: "User IDs" and "Profiles". The "User IDs" list contains one entry, "Jeff", which is selected. The "Profiles" list is empty. At the bottom of the window, there is a "Password:" label followed by the text "JeffPW".

**Figure 11-10**  
Local LU window with new user IDs and password

## Profiles

To create a security profile, select a user ID from the list of user IDs in the local LU window (see Figure 11-10). Then, choose Profile from the Create menu. The dialog box in Figure 11-11 appears. The example shows that a profile named *JeffProf1* has been created for local LU *LU1* and User ID *Jeff*.

The image shows a dialog box with the title "Enter profile for local LU 'LU1' and user ID 'Jeff':". It contains a "Profile:" label followed by a text field containing "JeffProf1". At the bottom of the dialog box are two buttons: "OK" and "Cancel".

**Figure 11-11**  
Creating a new profile

A profile is a security option for a local LU and a specific user ID. Each user ID may have multiple profiles. A profile has up to ten symbol-string type AE characters.

The new profile is displayed in the Profiles list of the local LU window when the user ID is selected, as shown in Figure 11-12.



**Local LU: LU1**

Local LU:

LU ID:

Net Name:

Net Qual:

Max Sessions:

LU Security: ☐ Yes ☒ No

Wait Time:

Max TPs:

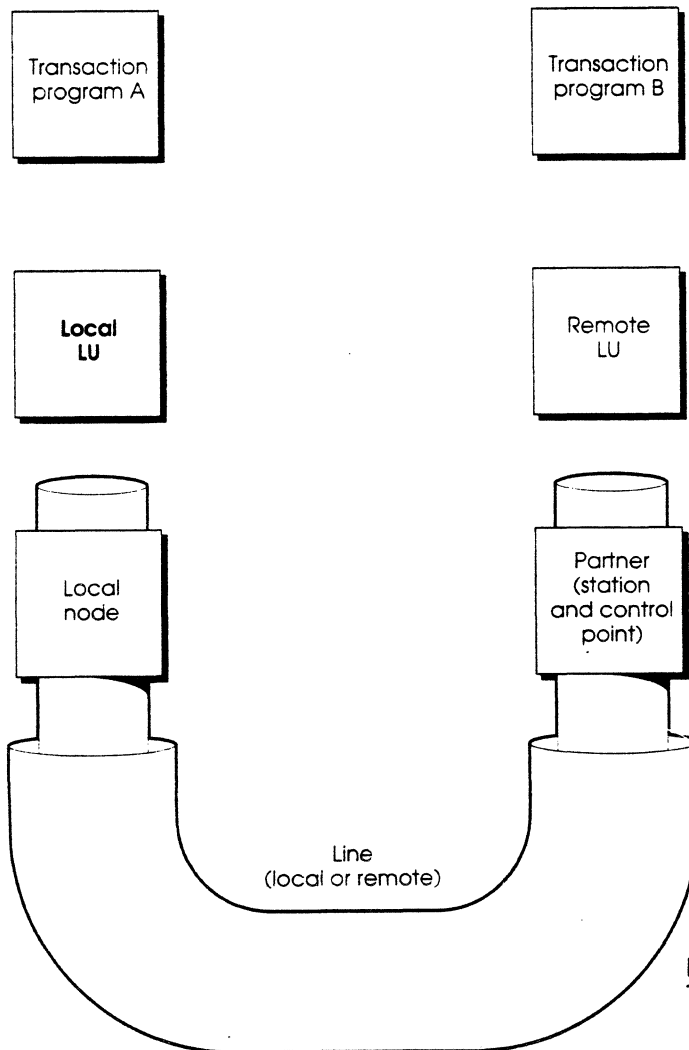
User IDs	Profiles
Jeff	JeffProf1

Password:

**Figure 11-12**

Local LU user IDs, profiles, and password

You have now specified the component settings for Local LU, LU ID, Max Sessions, Max TPs, User IDs, Password, and Profiles for your new local LU named *LU1*. Net Name, Net Qual, LU Security, and Wait Time are set to default values and may be changed by editing, as described in the later section on editing a local LU. Close the LU option window and return to the configuration file window. Figure 11-13 shows the local LU in its relationship to the other components.



**Figure 11-13**

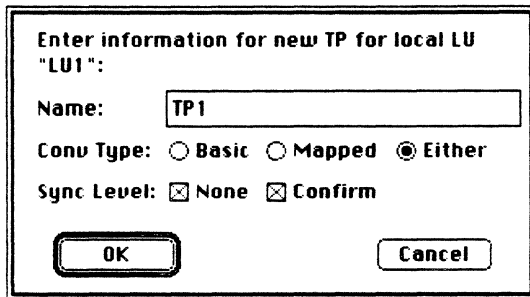
The local LU in relation to other components



---

## Creating transaction programs

One or more local LUs must be created before transaction programs (TPs) can be created. To create a transaction program for a local LU, select the appropriate local LU from the configuration file window Local LUs list (see Figure 11-5) and choose TP from the Create menu. The dialog box in Figure 11-14 appears. This example shows that a new TP named *TP1* is being created for the selected local LU *LU1*.



**Figure 11-14**  
Creating a transaction program for a local LU

### Name (Transaction program name)

The TP name must be unique for this local LU. TP names have up to 64 symbol-string type AE characters. The names *CNOS* and *ADMIN* are reserved by MacAPPC.

### Conv Type (Conversation type)

Conv Type specifies the conversation type allowed on an allocation request to start a program.

**Basic:** The TP may be allocated only with a basic conversation.

**Mapped:** The TP may be allocated only with a mapped conversation.

**Either:** The TP may be allocated with either a basic or a mapped conversation. The default setting is Either.

### Sync Level (Synchronization level)

**None:** A synchronization level of none can be used for this TP.

**Confirm:** A synchronization level of confirm can be used for this TP.

The default settings are both None and Confirm.



## Security Required

To select the security level for a TP, double-click the TP name in the Transaction Programs list of the configuration file window (see Figure 11-5). Alternatively, you may select a TP and choose Get Info from the File menu. The TP window in Figure 11-15 appears.

The TP window is a dialog box titled "TP: TP1". It contains the following fields and options:

- TP Name:** A text field containing "TP1".
- Local LU:** A text field containing "LU1".
- Net Name:** An empty text field.
- Status:** Radio buttons for ☒ Enable, ☐ Temp, and ☐ Perm.
- Conv Type:** Radio buttons for ☐ Basic, ☐ Map, and ☒ Either.
- Sync Level:** Check boxes for ☒ None and ☒ Confirm.
- PIP:** Radio buttons for ☐ Yes and ☒ No.
- PIP Count:** A text field containing "0".
- PIP Check:** Radio buttons for ☐ Yes and ☒ No.
- Data Mapping:** Radio buttons for ☐ Yes and ☒ No.
- FMH Data:** Radio buttons for ☐ Yes and ☒ No.
- Privilege:** Check boxes for ☒ None, ☐ CNOS, ☐ Session, ☐ Define, ☐ Disp, and ☐ Service.
- LUV:** Radio buttons for ☐ Yes and ☒ No.
- Security Required:** Radio buttons for ☒ None, ☐ Conv, ☐ User, ☐ Prof, and ☐ User/Prof.
- User ID:** A list box with an empty list and up/down arrow buttons.
- Profile:** A list box with an empty list and up/down arrow buttons.

**Figure 11-15**  
The TP window

Table 11-1 summarizes the security options available for a transaction program. Note that if None or Conv is selected for security required, user IDs and profiles cannot be created. If User is selected, only user IDs can be created. If Prof is selected, only profiles can be created. If User/Prof is selected, both user IDs and profiles can be created.

**Table 11-1**  
Security options

Security setting	User ID allowed	Profile allowed
None	No	No
Conv*	No	No
User†	Yes	No
Prof†	No	Yes
User/Prof†	Yes	Yes

\*Security checked at the LU level only.

†Security checked at the LU and TP levels.

To establish the security requirements for this TP, you select the security level to be implemented, a list of authorized user IDs, and a list of profiles from those created for the local LU. The following security options are available:

**None:** No security checking is performed for this TP. Note that if LU security is set to Yes for the local LU, security is checked at the local LU level, even if Security Required is None. The default setting is set to None.



**Conv (Conversation):** Conversation-level security is required for this TP. If security parameters are specified on a conversation allocation, they will be checked at the LU level only.

**Prof (Profile):** Resource-access-level security is required for this TP. The profile must match a profile defined on the resource-access authorization list.

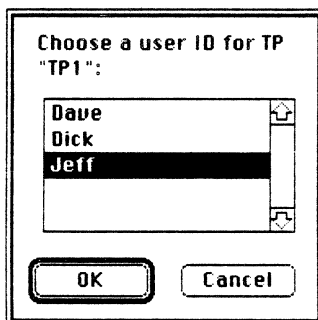
**User:** Resource-access-level security is required for this TP. The user must match a user ID defined on the resource-access authorization list.

**User/Prof (User ID and profile):** Resource-access level security is required for this TP. The user ID and profile must match both a user ID and a profile on the resource-access authorization list.

## User ID

User ID is a network security setting. A user ID is required only when specified on the resource-access authorization list. User IDs have up to ten symbol-string type AE characters. If you have selected a security level of None, Conv, or Prof, you cannot select User from the Create menu. If the selected security level is User or User/Prof, choose User from the Create menu. The dialog box in Figure 11-16 appears.

The dialog box displays the list of user IDs that were created for the local LU, but have not previously been chosen to be on the transaction program User ID list. Select a user ID for the TP. This example shows that the user ID named *Jeff* has been selected for TP *TP1*. If no user IDs appear on the list or if the desired ID does not appear on the list, you must first create the user ID required for the local LU. See the earlier section on creating a local LU.



**Figure 11-16**  
Creating a user ID for a TP

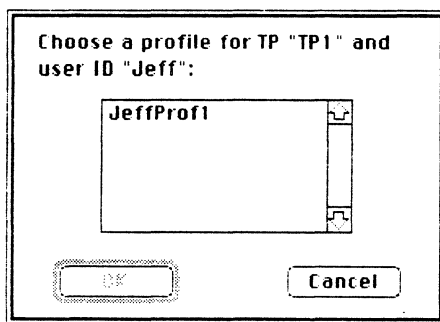
## Profile

Profile is a network security setting. A profile is required only when specified on the resource-access authorization list. Profiles have up to ten symbol-string type AE characters. A profile for a user can be selected only if a profile was created for the local LU and the transaction program Security Required setting is either Prof or User/Prof. If the Security Required setting is None, Conv, or User, you cannot choose Profile from the Create menu.



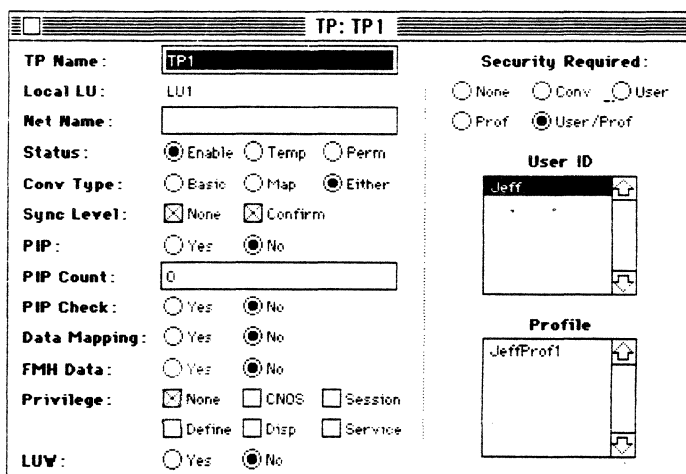
To choose a profile for a TP if the Security Required setting is Prof, first return to the TP window (see Figure 11-15) and choose Profile from the Create menu. A dialog box appears with a list of profiles that have been created for every user at the associated local LU, but have not previously been chosen to be on the transaction program Profile list for that user ID. If no profiles appear on the list or if the desired profile does not appear on the list, you must first create the profile required for the associated local LU and user ID. Then choose a profile from the list and click OK to add the profile to the Profile list in the TP window.

To choose a profile for a TP if the security required is User/Prof, first return to the TP window (see Figure 11-15) and select the associated user ID from the list displayed. Then, choose Profile from the Create menu. The dialog box in Figure 11-17 appears. The list contains the profiles that have been created for the related local LU for the user ID selected, but have not previously been chosen to be in the transaction program Profile list for that user ID. This example shows that the local LU profile named *JeffProf1* is available for selection as a profile for TP *TP1* and user ID *Jeff*. If no profiles appear on the list or if the desired profile does not appear on the list, you must first create the profile required for the associated local LU and user ID. Choose a profile from the list and click OK to add the profile to the Profile list in the TP window.



**Figure 11-17**  
Selecting a profile for a TP

Figure 11-18 shows the TP settings window for the example TP. Note that User/Prof has been selected for security required and that a profile is displayed for the selected user ID.

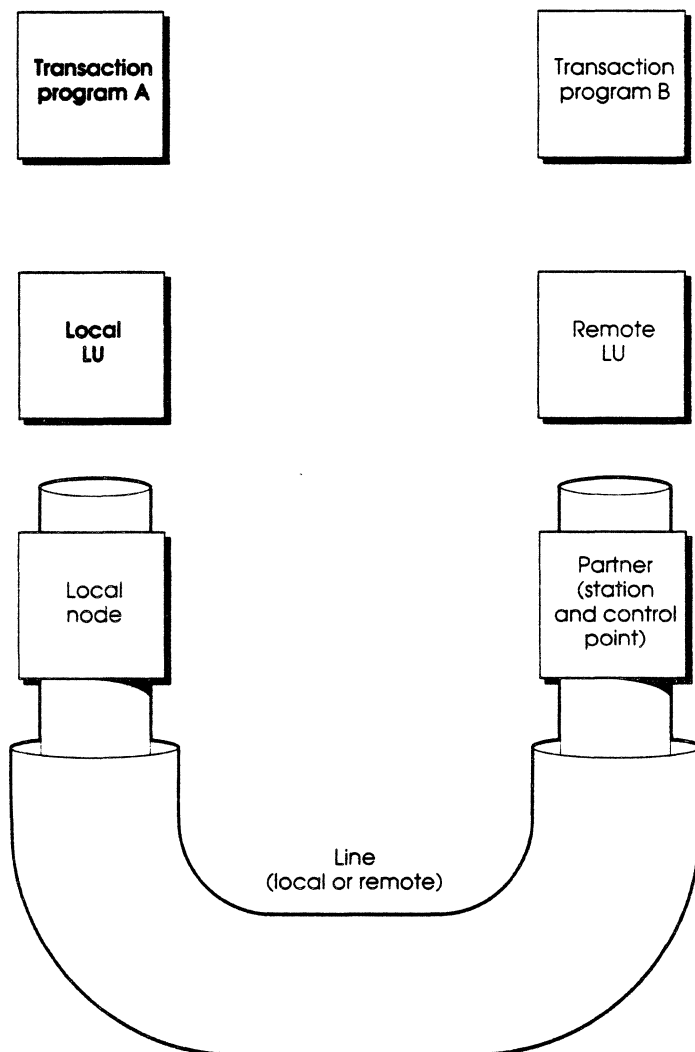


**Figure 11-18**  
TP window with User ID and Profile lists



You have now created a new transaction program and specified TP Name, Conv Type, Sync Level, Security Required, User ID, and Profile settings. Status, PIP, PIP Count, PIP Check, Data Mapping, FMH Data, Privilege, and LUW are set to default values and may be changed by editing, as described in the later section on editing a TP. Close the TP option window and return to the configuration file window.

Figure 11-19 shows the transaction program in its relationship to the other components.



**Figure 11-19**  
The transaction program in relation to other components

## Creating lines

Lines can be created at any time. For a local configuration, you may use the line named *LOCAL*, which is reserved by MacAPPC. For other configurations, you must create one or more lines before you can create partners, remote LUs, or modes. To create a line, choose Line from the Create menu. The dialog box in Figure 11-20 appears. This example shows that a new line named *LINE1* is being created.



Enter information for new line:

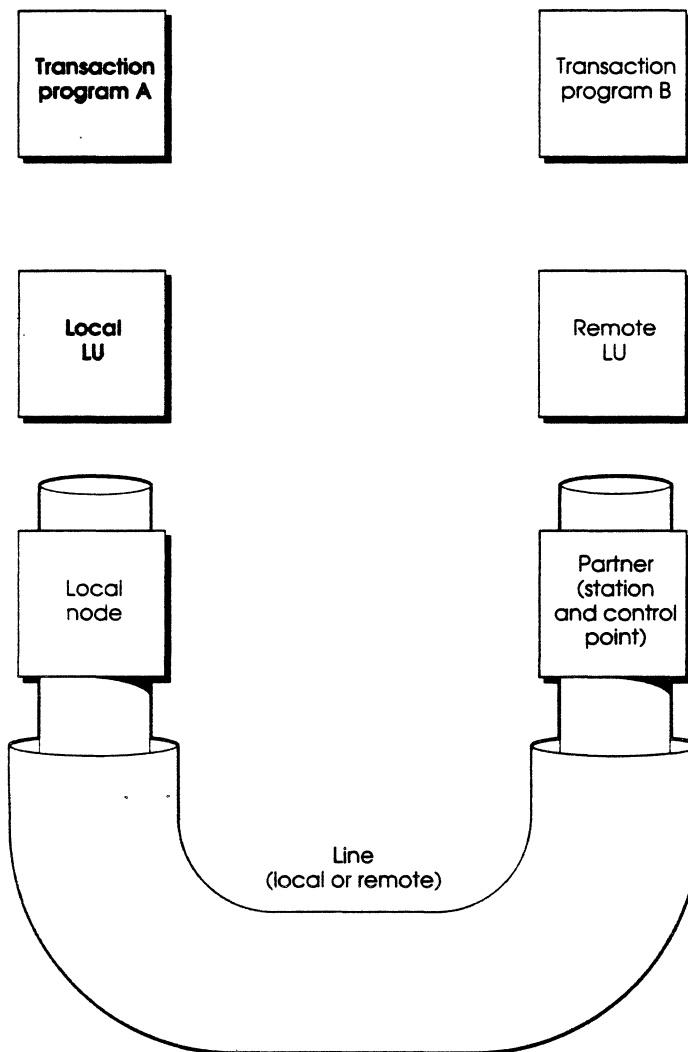
Name:

**Figure 11-20**  
Creating a line

### Name (Line name)

Line names have up to eight symbol-string type A characters. The name *LOCAL* is reserved by MacAPPC. You have now created a line and specified its name. Additional default settings have been assigned to this line that may be changed by editing, as described in the later section on editing a line.

Once you have created a line, you may then create a partner. If the line is multipoint or switched, you may create multiple partners. Figure 11-21 shows the line in its relationship to the other components.



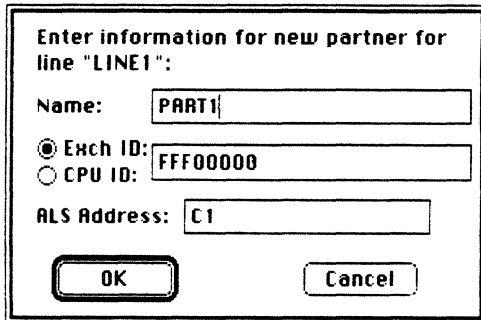
**Figure 11-21**  
The line in relation to other components



---

## Creating partners

After you have created one or more lines, you can create partners. A partner consists of both a station and control point. To create a partner, select the appropriate line from the Lines list in the configuration file window. Choose Partner from the Create menu. The dialog box in Figure 11-22 appears. This example shows that a new partner named *PART1* is being created for the selected line named *LINE1*.



Enter information for new partner for line "LINE1":

Name:

☒ Exch ID:

☐ CPU ID:

ALS Address:

**Figure 11-22**  
Creating a partner node

### Name (Partner name)

The partner name is the name by which the local LU can recognize the partner node being defined. Partner names have up to eight symbol-string type A characters. The name *LOCAL* is reserved by MacAPPC.

### Exch ID (Exchange ID) or CPU ID

You may specify either an exchange ID or a CPU ID; you may not specify both. The default setting is Exch ID.

### Exch ID (Exchange ID)

Exch ID is used in XID exchange at link establishment. Exchange IDs have exactly eight hex (0–F) characters representing a 4-byte binary number. They must contain the ID of the remote node if the node is a peer. In IBM implementations, the first three characters are set based on the product type (03A for the Displaywriter, 03E for the System/36, and so forth). The next five characters are user configurable to give unique exchange IDs throughout the network. The default setting is FFF00000.

If the partner is an IBM product, check the relevant IBM manuals for restrictions on the exchange ID value (the value that is received in bytes 2–5 of the XID). If the remote node is a peer, this field is supplied.

### CPU ID

CPU IDs have exactly twelve hex (0–F) characters. The first two characters (representing the first byte of the CPU ID) are the physical unit (PU) type (usually 05 for PU type 5). The remaining ten characters (representing the final 5 bytes) are an implementation-dependent binary identifier.

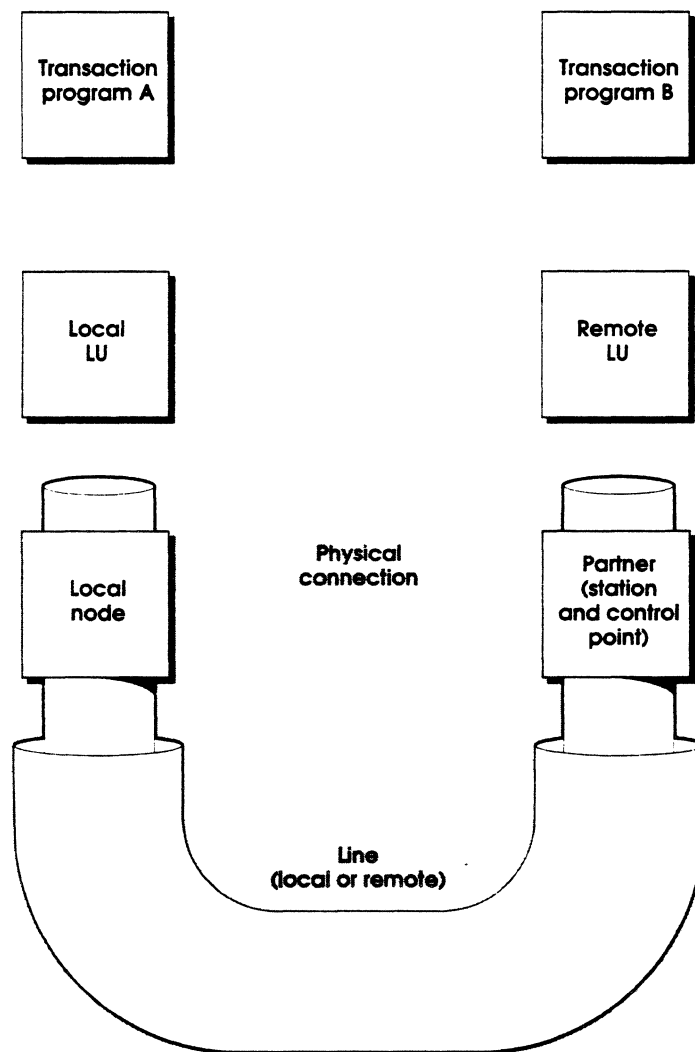


On a VTAM host, the 5 bytes are set to the subarea identifier of the host, specified by the SSCPID keyword in the VTAM ATCSTR definition. This value must match the value sent by the SSCP (system services control point) in bytes 3–8 of the Activate Physical Unit (ACTPU) request. If the remote PU is a host, this field is supplied.

### ALS Address (Adjacent-link-station address)

ALS Address specifies the adjacent-link-station address. Adjacent-link-station addresses have exactly two hex (0–F) characters. The default setting is C1. The values 00 and FF are invalid. For multipoint lines, each partner must have a unique ALS address.

You have now created a partner and specified its Name, Exch ID or CPU ID, and ALS Address settings. Additional default settings have been assigned to this partner that may be changed by editing, as described in the later section on editing a partner. Figure 11-23 shows the partner in its relationship to the other components.



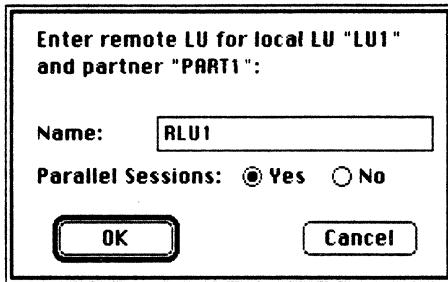
**Figure 11-23**  
The partner in relation to other components



---

## Creating remote LUs

You must create one or more local LUs, lines, and partners before you can create a remote LU. To create a remote LU, select the appropriate local LU, line, and partner from the lists displayed in the configuration file window. A box is drawn around each component selected. Choose Remote LU from the Create menu. The dialog box in Figure 11-24 appears. This example shows that a remote LU named *RLU1* is being created for the local LU *LU1* and partner *PART1*.



**Figure 11-24**  
Creating a remote LU

### Name (Remote LU name)

The remote LU name is the name by which the local LU can recognize the remote LU. Remote LU names have up to eight symbol-string type A characters.

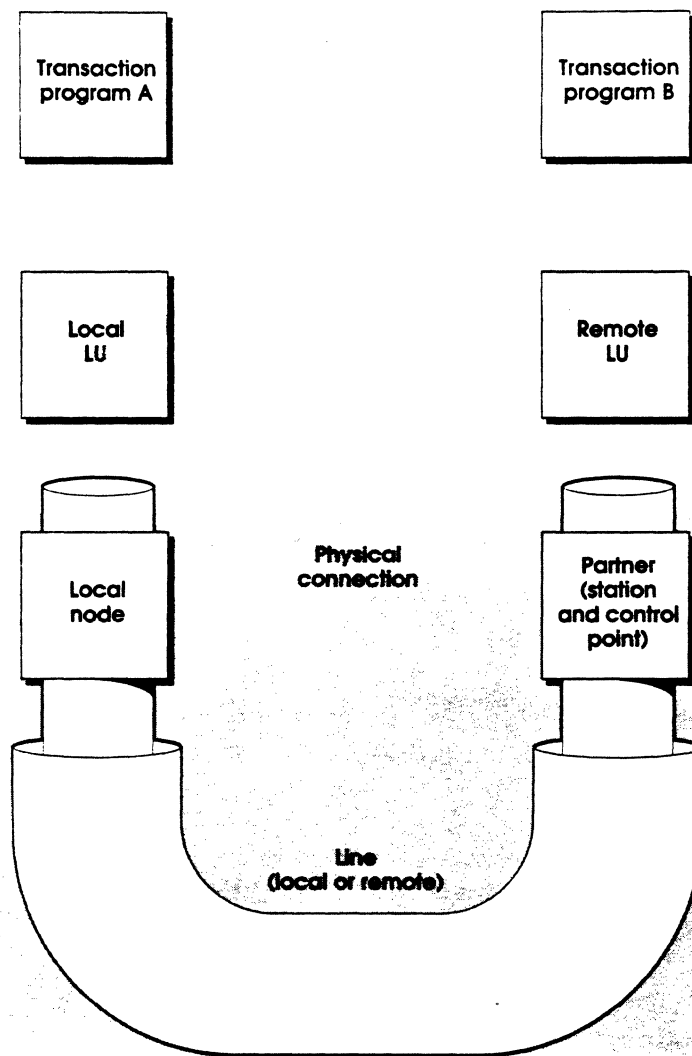
### Parallel Sessions

Parallel Sessions specifies whether or not two LUs may connect to each other by means of multiple LU-LU sessions over a single mode. Yes indicates that parallel sessions are supported; No indicates that only single sessions are supported. The default setting is Yes.

You have now created a remote LU and specified its Name and Parallel Sessions settings. Additional default settings have been assigned to this remote LU that may be changed by editing, as described in the later section on editing a remote LU.



Figure 11-25 shows the remote LU in its relationship to the other components.



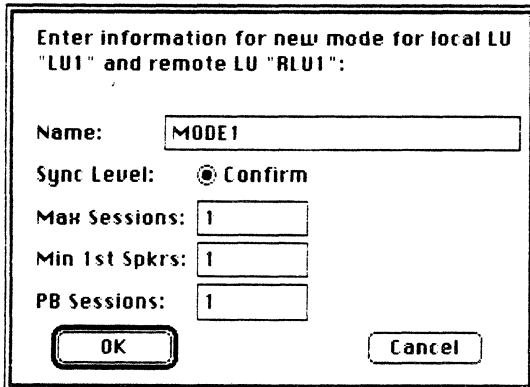
**Figure 11-25**  
The remote LU in relation to other components



---

## Creating modes

To create a mode, select the appropriate local LU, line, partner, and remote LU from the lists displayed in the configuration file window. A box is drawn around each component selected. Choose Mode from the Create menu. The dialog box in Figure 11-26 appears. This example shows that a new mode named *MODE1* is being created for the local LU *LU1* and the remote LU *RLU1*.



The dialog box contains the following fields and controls:

- Name:** A text field containing the value *MODE1*.
- Sync Level:** A radio button labeled *Confirm* is selected.
- Max Sessions:** A text field containing the value *1*.
- Min 1st Spkrs:** A text field containing the value *1*.
- PB Sessions:** A text field containing the value *1*.
- Buttons:** *OK* and *Cancel* buttons are located at the bottom.

**Figure 11-26**  
Creating a mode

### Name (Mode name)

The mode name specifies the name of the group of sessions that will have the characteristics defined by the mode's options. Mode names have up to eight symbol-string type A characters. The name *SNASVCMG* is a reserved mode name and may not be used.

### Sync Level (Synchronization level)

Sync Level specifies the synchronization levels that conversations using sessions over this mode may use. Confirm is the only synchronization level currently supported. The default setting is Confirm.

### Max Sessions (Maximum number of sessions)

Max Sessions specifies the maximum number of sessions that can be active at a given time for this mode. The valid range is 1 to 254. If a setting greater than 1 is specified, parallel sessions must be supported by the remote LU. The default setting is 1.

### Min 1st Spkrs (Minimum number of first speakers)

Min 1st Spkrs specifies the minimum number of first-speaker sessions for this mode. The valid range is 1 to the value specified for Max Sessions. The default setting is 1.

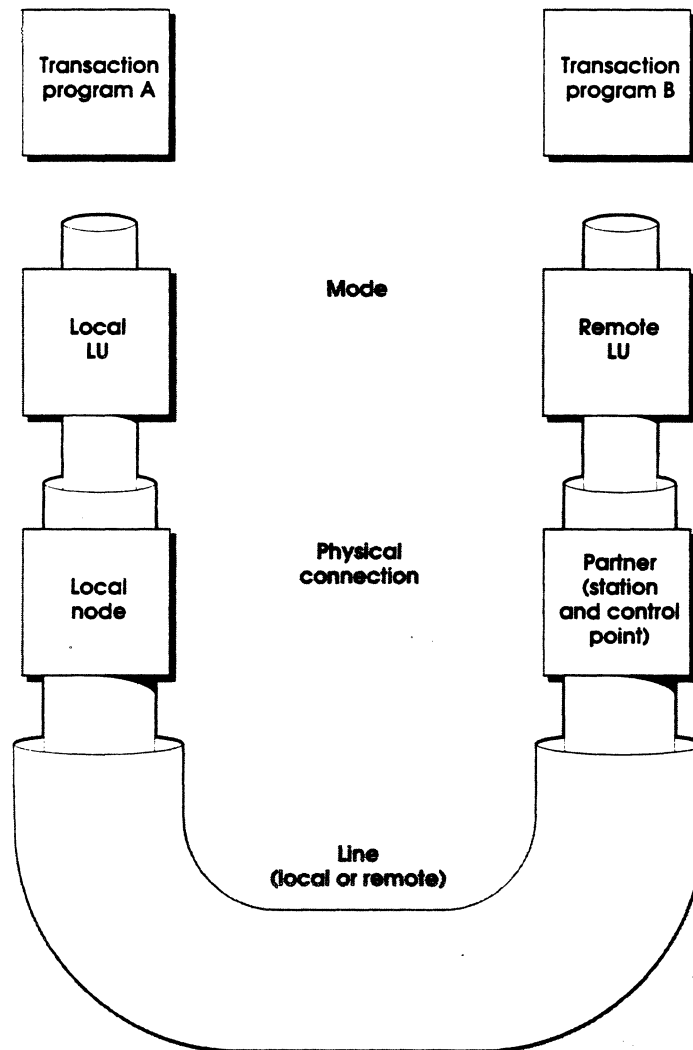


### PB Sessions (Number of prebound sessions)

PB Sessions specifies the number of first-speaker sessions that are automatically activated (prebound) when session limits are initialized. This setting corresponds to the Contention Winner Auto-Activation Limit. The valid range is 1 to the value specified for Min 1st Spkrs. The default setting is 1.

You have now created a mode and specified its Name, Sync Level, Max Sessions, Min 1st Spkrs, and PB Sessions settings. Additional default settings have been assigned to this mode that may be changed by editing, as described in the later section on editing a mode.

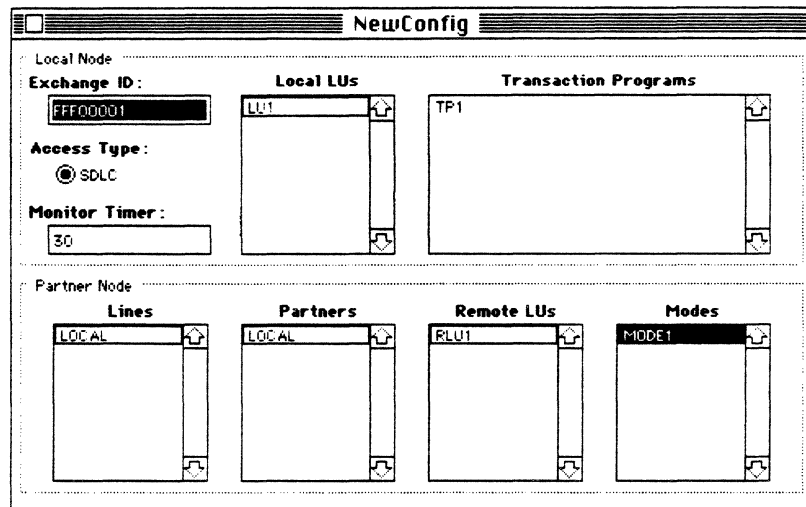
Figure 11-27 shows a mode in its relationship to the other components.



**Figure 11-27**  
The mode in relation to other components



The network node components have now been created and their relationships to each other are established. The configuration file window for our example node is shown in Figure 11-28. The component settings must now be edited to complete a detailed description of each component, as required for a full characterization of the node's operational characteristics.



**Figure 11-28**  
The configuration file window for the example network node

## Editing network components

The previous section showed you how to create network components; for most components, you simply assign a name, and specify settings for that resource. For some settings, default settings were automatically set by the Configuration program for each component. Once network components have been created, you can edit the full range of settings that define each one. Often you may want to use the same setting for a component option across multiple instances of network components. If this is the case, it is easier to edit the default settings before creating them instead of editing each component individually later. There is a section on editing defaults later in this chapter.

### Editing a local LU

To edit a local LU, first return to the configuration file window (see Figure 11-28) and double-click the local LU to be edited from the Local LUs list. The window in Figure 11-29 appears. This example shows a local LU named *LU1* that has been selected for editing.

Use this window to edit the settings of the selected local LU that were specified when the local LU was created.



**Figure 11-29**  
Editing a local LU

## Local LU

Local LU is the name used to associate mode, remote LU, and transaction program definitions with a particular local LU. Local LU names have up to eight symbol-string type A characters.

## LU ID

LU ID is the local LU ID number. The LU ID setting corresponds to the destination address field (DAF). Each local LU must have a unique LU ID. The valid range is 1 to 254. The default setting is 1.

## Net Name (Local LU network name)

Net Name is the name by which the local LU is known throughout the network. Local LU network names have up to eight symbol-string type A characters. If not specified (left blank), the local LU name is used as the default value for the network name.

## Net Qual (Local LU network qualifier name)

Net Qual is an optional field name that may be defined for cross-network communications to give uniqueness to the local LU network name. Network qualifiers have up to eight symbol-string type A characters.

## Max Sessions (Maximum number of sessions)

Max Sess is the maximum number of sessions that can be active at the local LU at one time. Session limits are currently maintained only at a mode level. The valid range is 0 to 254. Specifying 0 indicates that this setting should be ignored. The default setting is 1.



## LU Security

LU Security indicates whether or not conversation-level security is enabled. If it is, the user ID and the password are checked at the conversation level, regardless of the TP-defined security. Yes specifies that conversation-level security is enabled for the local LU; No specifies that it is not. The default setting is No.

## Wait Time

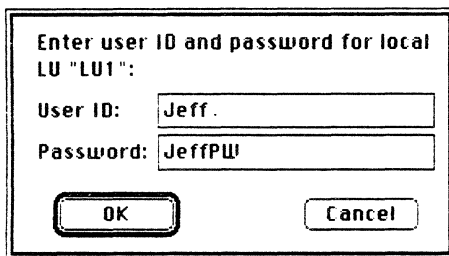
Wait Time specifies the amount of time in seconds the LU waits for routine completion. The valid range is 1 to 3600. The default setting is 60.

## Max TPs (Maximum number of transaction programs)

Max TPs specifies the maximum number of local transaction programs that can be attached to the local LU at one time. The valid range is 1 to 255. The default setting is 5.

## User IDs

User IDs lists the user IDs that have been created for the local LU. User IDs have up to ten symbol-string type AE characters. To edit a user ID, double-click it. The dialog box in Figure 11-30 appears.

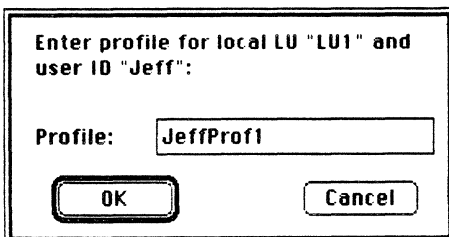


A dialog box titled "Enter user ID and password for local LU 'LU1':" with two input fields. The "User ID:" field contains the text "Jeff." and the "Password:" field contains the text "JeffPW". At the bottom are two buttons: "OK" and "Cancel".

**Figure 11-30**  
Editing a user ID

## Profiles

Profiles lists the profiles that have been created for the local LU. Profiles have up to ten symbol-string type AE characters. To edit a profile, select its associated user ID, then double-click the profile name to be edited. The dialog box in Figure 11-31 appears.



A dialog box titled "Enter profile for local LU 'LU1' and user ID 'Jeff':" with one input field. The "Profile:" field contains the text "JeffProf1". At the bottom are two buttons: "OK" and "Cancel".

**Figure 11-31**  
Editing a profile



## Password

Password shows the password for a selected user ID. Passwords have up to ten symbol-string type AE characters. To display a user ID's password, click the user ID from the User IDs list in the local LU window. To edit a password, double-click the user ID from the User IDs list. The dialog box in Figure 11-30 appears.

---

## Editing a transaction program

To edit a transaction program, first return to the configuration file window (see Figure 11-28) and click the local LU associated with the transaction program. Then, double-click the name of the transaction program to be edited from the transaction program list. The window in Figure 11-32 appears. This example shows a TP named *TP1* that has been selected for editing.

Use this window to edit the settings that were specified when the TP was created.

The dialog box is titled "TP: TP1". It contains the following fields and options:

- TP Name:** A text field containing "TP1".
- Local LU:** A text field containing "LU1".
- Net Name:** An empty text field.
- Status:** Radio buttons for ☒ Enable, ☐ Temp, and ☐ Perm.
- Conv Type:** Radio buttons for ☐ Basic, ☐ Map, and ☒ Either.
- Sync Level:** Check boxes for ☒ None and ☒ Confirm.
- PIP:** Radio buttons for ☐ Yes and ☒ No.
- PIP Count:** A text field containing "0".
- PIP Check:** Radio buttons for ☐ Yes and ☒ No.
- Data Mapping:** Radio buttons for ☐ Yes and ☒ No.
- FMH Data:** Radio buttons for ☐ Yes and ☒ No.
- Privilege:** Check boxes for ☒ None, ☐ CNOS, ☐ Session, ☐ Define, ☐ Disp, and ☐ Service.
- LUV:** Radio buttons for ☐ Yes and ☒ No.
- Security Required:** Radio buttons for ☐ None, ☐ Conv, ☐ User, ☐ Prof, and ☒ User/Prof.
- User ID:** A list box containing "Jeff".
- Profile:** A list box containing "JeffProf1".

**Figure 11-32**  
Editing a transaction program

### TP Name

The TP Name must be unique for this local LU. TP names have up to 64 symbol-string type AE characters. The names *CNOS* and *ADMIN* are reserved by MacAPPC.

### Local LU

The local LU name cannot be modified by editing a TP. See the section on editing local LUs.

### Net Name (Transaction program network name)

Net Name is the name by which the TP is known throughout the network. TP network names have up to 64 hex (0–F) characters. If not specified (left blank), the TP name is used as the network name.



## Status

Status specifies the response the LU makes when an allocation request is received that designates this TP. By changing this setting, a program may be temporarily or permanently removed from network availability.

**Enable (Enabled):** The TP is enabled. The default setting is Enabled.

**Temp (Temporary):** The TP is temporarily disabled.

**Perm (Permanent):** The TP is permanently disabled.

## Conv Type (Conversation type)

Conv Type specifies the conversation type allowed on an allocation request to start the program.

**Basic (Basic conversation type):** The TP may be allocated only with a basic conversation.

**Map (Mapped conversation type):** The TP may be allocated only with a mapped conversation.

**Either (Either conversation type):** The TP may be allocated with either a basic or a mapped conversation. The default setting is Either.

## Sync Level (Synchronization level)

Sync Level specifies the synchronization allowed on an allocation request to start the program.

**None:** The synchronization level is none.

**Confirm:** The synchronization level is confirm.

The default settings are both None and Confirm.

## PIP (Program initialization parameters)

PIP indicates whether or not this transaction program supports program initialization parameters. Yes indicates that PIP is supported; No indicates that it is not. The default setting is No.

## PIP Count

PIP Count is the program initialization parameters count. It specifies the number of PIP subfields that may be required in order to start this transaction program. The valid range is from 0 to 256. The default setting is 0.

## PIP Check

PIP Check indicates whether or not this transaction program supports program initialization parameter checking. No indicates that an allocation request for the TP is not rejected, even if the number of PIP subfields in the PIP Count do not match the number supplied on the request. Yes indicates that an allocation request for the transaction program is rejected if the number of PIP subfields in the PIP Count do not match the number supplied on the request. The default setting is No.



## Data Mapping

Data Mapping indicates whether or not this transaction program is provided with data mapping support. Yes indicates that data mapping is supported; No indicates that it is not. The default setting is No.

## FMH Data

FMH Data indicates whether or not functional management header (FMH) data can be transmitted and received by this transaction program. Yes indicates that the transaction program can transmit and receive this information; No indicates that it cannot. The default setting is No.

## Privilege

Privilege specifies the type of control operator routines this transaction program may issue.

**None:** The TP is not allowed to use routines that require a privilege to do so. The default setting is None.

**CNOS:** The TP is allowed to issue change-number-of-session CNOS routines.

**Session:** The TP is allowed to issue session control routines.

**Define:** The TP is allowed to issue definition routines.

**Disp (Display):** The TP is allowed to issue display routines.

**Service:** The TP is allowed to issue an allocation routine with the TP name designating a service transaction program, such as SNADS or DIA.

## LUW (logical unit of work)

LUW indicates whether or not this transaction program assigns logical-unit-of-work IDs for each transaction issued by this program. Yes specifies that the TP assigns LUW IDs; No specifies that it does not. The default setting is No.

## Security Required (Security level that is required)

Security Required specifies the security verification allowed on an allocation request to start the program.

**None:** No security checking is performed for this TP. Note that if LU security is set to Yes for the local LU, security is checked at the local LU, even if Security Required is None. The default setting is None.

**Conv (Conversation-level security):** Conversation-level security is specified. If security settings are specified on a conversation allocation, they are checked at the LU level only.

**Prof (Profile):** Resource-access-level security is required for this TP. The profile must match a profile defined on the resource-access authorization list.

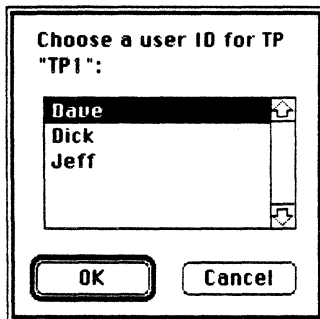
**User (User ID):** Resource-access-level security is required for this TP. The user ID must match a user ID defined on the resource-access authorization list.



**User/Prof (User ID and profile):** Resource-access-level security is required. The user ID and profile must match both a user ID and a profile on the resource-access authorization list.

## User ID

User ID is a network security setting. A user ID must be included if the resource-access authorization list requires a user ID. User IDs have up to ten symbol-string type AE characters. Nothing appears in the User ID list unless User or User/Prof is selected for Security Required. To delete a user ID, select it, and then choose Clear from the Edit menu. To add a new user ID, choose User from the Create menu. The dialog box in Figure 11-33 appears. The list contains the users that have been created for the associated local LU, but have not previously been chosen to be on the transaction program User ID list. Select a user ID, and then click OK to add the user ID to the User ID list in the TP window.



**Figure 11-33**  
Creating a User ID

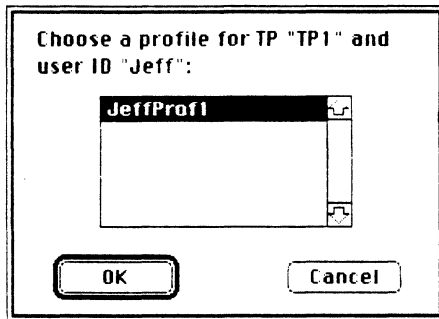
## Profile

Profile is a network security setting. A profile must be included if the resource-access authorization list requires a profile. Profiles have up to ten symbol-string type AE characters. A profile for a user can be selected only if a profile was created for the local LU and the transaction program Security Required setting is either Prof or User/Prof. If the selected Security Required setting is None, Conv, or User, you cannot choose Profile from the Create menu.

To choose a profile for a TP if the Security Required setting is Prof, first return to the TP window (see Figure 11-32) and choose Profile from the Create menu. A dialog box with a list of all the profiles that have been created for the associated local LU, but have not previously been chosen to be on the transaction program Profile list. If no profiles appear on the list or if the desired profile does not appear on the list, you must first create the profile required for the associated local LU. Then, choose a profile from the list and click OK to add the profile to the Profile list in the TP window.



To choose a profile for a TP if the security required is User/Prof, first return to the TP window (see Figure 11-32) and select the user ID from the list displayed. Then, choose Profile from the Create menu. The dialog box in Figure 11-34 appears. The list contains the profiles that have been created for the associated local LU for the user ID selected, but have not previously been chosen to be in the transaction program Profile list for that user ID. This example shows that the local LU profile named *JeffProf1* is available for selection as a profile for TP *TP1* and user ID *Jeff*. If no profiles appear on the list or if the desired profile does not appear on the list, you must first create the profile required for the associated local LU and user ID. Choose a profile from the list and click OK to add the profile to the Profile list in the TP window.



**Figure 11-34**  
Creating a profile

## Editing a line

To edit a line, first return to the configuration file window (see Figure 11-28) and double-click the line to be edited from the Lines list. The window in Figure 11-35 appears. This example shows that a line named *LINE1* has been selected for editing.

Use this window to edit the settings that were specified when the line was created.

 A window titled "Line: LINE1". It contains several configuration fields:
 

- Line Name:** A text box containing "LINE1".
- Line Type:** A text box containing "SDLC".
- Line Number:** Radio buttons for 1, 2, 3, and 4. Radio button 1 is selected.
- Role Type:** Radio buttons for Prim, Sec, and Negot. Radio button Negot is selected.
- Connect Type:** Radio buttons for Lease, Multi, and Switch. Radio button Lease is selected.
- Max BTU:** A text box containing "265".
- Line Speed:** Radio buttons for 300, 1200, 2400, 4800, 9600, and 19200. Radio button 9600 is selected.
- Max Retries:** A text box containing "3".
- Idle Time:** A text box containing "800".
- NP Recv Time:** A text box containing "10000".
- Max I-Frames:** A text box containing "7".
- NRZI Support:** Radio buttons for NRZ and NRZI. Radio button NRZ is selected.
- Duplex Type:** Radio buttons for Half and Full. Radio button Half is selected.

**Figure 11-35**  
Editing a line



## Line Name

Line name specifies the name of the line to be edited. The name can have up to eight symbol-string type A characters. The name *LOCAL* is reserved by MacAPPC.

## Line Type

Line Type is based on the access type of the local node. Synchronous Data Link Control (SDLC) is the only line type that is currently available.

## Line Number

Line Number specifies the serial port number to which the line is connected. The line number may be 1, 2, 3, or 4. The default setting is 1.

## Role Type

Role Type specifies whether or not the synchronous data link control role is primary, secondary, or negotiable.

**Prim (Primary):** The SDLC role is primary.

**Sec (Secondary):** The SDLC role is secondary.

**Negot (Negotiable):** The SDLC role is negotiable. The default setting is Negot.

## Connect Type (Connection type)

**Lease (Leased):** A leased connection is a directly connected line. The default setting is Leased.

**Multi (Multipoint):** A multipoint connection is a party-line in which several users share the same line.

**Switch (Switched):** A switched connection is established when required and broken when a session is completed.

## Max BTU (Maximum basic transmission unit length)

Max BTU specifies the maximum basic transmission unit length that can be received on this line. This value is exchanged with the partner at link initiation. The valid range is 128 to 265. The default setting is 265.

## Line Speed

Line Speed specifies the transmission line speed in bits per second (bps). Valid settings are 300, 1200, 2400, 4800, 9600, and 19200. The default setting is 9600.

## Max Retries (Maximum number of retries)

Max Retries specifies the maximum number of times a frame is retransmitted after SDLC procedures have detected a discrepancy. When the number is exceeded, SDLC reports the problem to a higher level of SNA for resolution. The valid range is 1 to 30. The default setting is 3.



## Idle Time

Idle Time specifies the amount of time in milliseconds that can elapse without a response before a primary link station's initiate recovery action. The valid range is 100 to 10000 (0.100 to 10.000 seconds). The default setting is 800 (0.800 seconds).

## NP Recv Time (Nonproductive receive time)

NP Recv Time specifies the amount of time that can elapse before the primary link station reports to a higher level of SNA that a nonproductive receive condition exists for resolution. The valid range is 1000 to 30000 (1.000 to 30.000 seconds). The default setting is 10000 (10.000 seconds).

## Max I-Frames (Maximum number of I-frames)

Max I-Frames specifies the maximum number of I-frames that can be sent before polling resumes. The valid range is 1 to 7. The default setting is 7.

## NRZI Support

NRZI Support specifies the type of transmission encoding method to be used when sending signals over this line.

**NRZ:** A nonreturn-to-zero (NRZ) encoding method is used. The default setting is NRZ.

**NRZI:** A nonreturn-to-zero-inverted (NRZI) encoding method is used.

## Duplex Type

Duplex Type specifies whether data can be sent in one or both directions without turning the line around.

**Half:** The transmissions are half-duplex. Data can be sent in only one direction and then the line must be turned around. The default setting is Half.

**Full:** The transmissions are full-duplex. Data can be sent and received without turning the line around.

---

## Editing a partner

To edit a partner, first return to the configuration file window (see Figure 11-28), and double-click the partner to be edited from the partners list. The window shown in Figure 11-36 appears. This example shows that a partner named *PART1* has been selected for editing. Use this window to edit the settings that were specified when the partner was created. A partner represents both a control point and a station.



Partner: PART1

Partner Name: PART1

Line Name: LINE1

☒ Exch ID: FFFD0000

☐ CPU ID:

ALS Address: C1

Phone Number:

**Figure 11-36**  
Editing a partner (control point and station)

## Partner Name

Partner Name is the name by which the local LU can recognize the partner being defined. Partner names have up to eight symbol-string type A characters. The name *LOCAL* is reserved by MacAPPC.

## Line Name

Line Name cannot be changed by editing a partner. See the earlier section on editing a line.

## Exch ID (Exchange ID) or CPU ID

Specify either an exchange ID or a CPU ID; you may not specify both.

### Exch ID

Exch ID is used in XID exchange at link establishment. Exchange IDs have exactly eight hex (0–F) characters. They must contain the ID of the remote node if the node is a peer. In IBM implementations, the first three characters are set based on the product type (03A for the Displaywriter, 03E for the System/36, and so forth). The next five characters are user configurable to give unique exchange IDs throughout the network.

If the partner is an IBM product, check the relevant IBM manuals for restrictions on the exchange ID value. This is the value that is received in bytes 2–5 of the XID. If the remote node is a peer, this field is supplied.

### CPU ID

CPU IDs have exactly twelve hex (0–F) characters. The first character of a CPU ID must be a zero. The first two characters (representing the first byte of the parameter CPU ID) are the physical unit (PU) type (usually 05, for PU type 5). The remaining ten characters (representing the final five bytes) are an implementation-dependent binary identifier. On a VTAM host, the five bytes are set to the subarea identifier of the host, specified by the SSCPID keyword in the VTAM ATCSTR definition. This value must match the value sent by the SSCP (system services control point) in bytes 3–8 of the Activate Physical Unit (ACTPU) request. If the remote PU is a host, this field is supplied.



## ALS Address

ALS Address is the adjacent-link-station address. Adjacent-link-station addresses have exactly two hex (0–F) characters. The default setting is C1. The values 00 and FF are invalid. For multipoint lines, each partner must have a unique ALS address.

## Phone Number

Phone Number provides reference information to a network user. Phone numbers are symbol-strings of up to 20 characters of type AE format. If the connect type of the line is switched and logging is enabled, this phone number is included in a log message when you activate that station.

---

## Editing a remote LU

To edit a remote LU, first return to the configuration file window (see Figure 11-28) and double-click the remote LU to be edited from the Remote LUs list. The window in Figure 11-37 appears. This example shows that a remote LU named *RLU1* has been selected for editing.

Use this window to edit the settings for the selected remote LU that were specified when the remote LU was created.

Remote LU: RLU1	
Remote LU:	RLU1
Local LU:	LUI
Net Name:	
Net Qual:	
CP Name:	PART1
Init Q Req:	<input checked="" type="radio"/> Yes <input type="radio"/> No
Parallel Sess:	<input checked="" type="radio"/> Yes <input type="radio"/> No
CNOS ALS:	PART1
Password:	
Lcl Sec:	<input checked="" type="radio"/> None <input type="radio"/> User <input type="radio"/> Verif

**Figure 11-37**  
Editing a remote LU

## Remote LU

Remote LU specifies the name by which the local LU can recognize the remote LU. Remote LU names have up to eight symbol-string type A characters.

## Local LU

Local LU specifies the name of the local LU for which the remote LU is being defined. A local LU cannot be changed by editing a remote LU. See the earlier section on creating a local LU.



### **Net Name (Remote LU network name)**

Net Name specifies the network name by which the remote LU is known throughout the network. If not specified (left blank), the remote LU name is used as the network name. Remote LU network names have up to eight symbol-string type A characters.

### **Net Qual (Remote LU network qualifier)**

Net Qual is an optional field that may be defined for cross-network communications to give uniqueness to the network name. Network qualifiers have up to eight symbol-string type A characters.

### **CP Name (Control point or partner name)**

CP Name specifies the name of the control point or partner where the remote LU is located. A CP Name cannot be changed by editing a remote LU. See the earlier section on creating a partner.

### **Init Q Req (Queue session-initiation requests)**

Init Q Req specifies whether or not session requests are queued by system services control point (SSCP). Yes indicates that SSCP queues session-initiation requests if the remote LU is not able to accept them; No indicates that it does not. The default setting is Yes.

### **Parallel Sess (Parallel sessions)**

Parallel Sess specifies whether or not the remote LU supports parallel sessions. This information is used during session activation by BIND requests and responses. Yes indicates that this function is supported; No indicates that it is not. The default setting is Yes.

### **CNOS ALS (CNOS adjacent-link-station name)**

CNOS ALS specifies the name of the adjacent link station (ALS) that handles change-number-of-sessions (CNOS) requests. If the local LU and remote LU are in the same node, this name is LOCAL. Otherwise, it's the name of a previously defined adjacent-link-station control block. If parallel sessions are supported, a name must be specified. In general, this is the partner name. CNOS adjacent-link-station names have up to 8 symbol-string type A characters.

### **Password**

Password is defined for the local LU by the remote LU. It is used to provide session-level verification during session activation. Passwords have up to 16 hex (0-F) characters.



## Lcl Sec (Local security)

Lcl Sec specifies the access-security information that the local LU accepts for the remote LU being defined.

**None:** No security is accepted. The default setting is None.

**User:** Access-security information is accepted. The already-verified indication is not accepted.

**Verif (Verified):** Access-security information is accepted. The already-verified indication is accepted.

---

## Editing a mode

To edit a mode, first return to the configuration file window (see Figure 11-28) and select the mode to be edited from the mode list. Then, double-click the selected mode. The window in Figure 11-38 appears. This example shows that a mode named *MODE1* has been selected for editing.

Use this window to edit the settings that were specified when the mode was created.

<b>Mode Name:</b> MODE1		<b>Sync Level:</b> <input checked="" type="radio"/> Confirm
<b>Local LU:</b> LU1	<b>Sess Reinit:</b> <input checked="" type="radio"/> None <input type="radio"/> Oper <input type="radio"/> Prim	
<b>Remote LU:</b> RLU1	<input type="radio"/> Sec <input type="radio"/> Either	
<b>Adj Station:</b> PART1	<b>Max Sessions:</b>	1
<b>Send Pacing:</b> 3	<b>Min 1st Spkrs:</b>	1
<b>Recv Pacing:</b> 3	<b>PB Sessions:</b>	1
<b>Max RU UB:</b> 1024	<b>Queue Binds:</b>	<input checked="" type="radio"/> Yes <input type="radio"/> No
<b>Max RU LB:</b> 256	<b>Blank Mode:</b>	<input type="radio"/> Yes <input checked="" type="radio"/> No

**Figure 11-38**  
Editing a mode

### Mode Name

Mode Name specifies the name of the group of sessions that have the characteristics defined by the mode's settings. Mode names have up to eight symbol-string type A characters. The name *SNASVCMG* is a reserved mode name and may not be used.

### Local LU

Local LU specifies the name of the local LU for which the mode is being defined. A local LU name cannot be changed by editing a mode. See the earlier section on creating a local LU.

### Remote LU

Remote LU specifies the name of the remote LU for which a mode is being defined. A remote LU name cannot be changed by editing a mode. See the earlier section on creating a remote LU.



### **Adj Station (Adjacent station or partner name)**

Adj Station specifies the name for the adjacent station or partner. An adjacent station or partner name cannot be changed by editing a mode. See the earlier section on creating a partner.

### **Send Pacing**

Send Pacing specifies the number of requests the local LU can send before receiving a pacing response. This is used to determine the send window size during session activation. The valid range is 0 to 21. The default setting is 3.

### **Recv Pacing (Receive pacing)**

Recv Pacing specifies the number of requests the local LU can receive. This is used to determine the receive window size during session activation. The valid range is 0 to 21. The default setting is 3.

### **Max RU UB (Maximum request/response unit upper bound)**

Max RU UB specifies the upper-bound of the maximum request/response unit (RU) size that can be sent or received across this mode. This field is used in conjunction with the maximum RU lower-bound setting to determine the maximum RU size possible during session activation. The valid range is 256 to 4096. The default setting is 1024.

### **Max RU LB (Maximum request/response unit lower bound)**

Max RU LB specifies the lower-bound of the maximum request/response unit (RU) size that can be sent or received across this mode. This field is used in conjunction with the maximum RU upper-bound setting to determine the maximum RU size possible during session activation. The valid range is 8 to 256. The default setting is 256.

### **Sync Level (Synchronization level)**

Sync Level specifies the synchronization levels that conversations using sessions over this mode may use. Confirm is the only synchronization level currently supported.

### **Session Reinit (Session reinitiation)**

Session Reinit specifies the responsibility for a single-session reinitiation. This can only be specified when the partner does not support parallel sessions.

**None:** There is no single-session reinitiation. The default setting is None.

**Oper (Operator):** An operator from either the local or remote LU attempts session reinitiation. Neither LU attempts automatic reinitiation.

**Prim (Primary LU):** The primary LU automatically attempts the reinitiation.

**Sec (Secondary LU):** The secondary LU automatically attempts the reinitiation

**Either (Either primary or secondary LU):** Either the primary or the secondary LU automatically attempts the reinitiation.



### **Max Sessions (Maximum number of sessions)**

Max Sessions specifies the maximum number of sessions that can be active at a given time for this mode. The valid range is 1 to 254. If a setting greater than 1 is specified, parallel sessions must be supported by the remote LU. The default setting is 1.

### **Min 1st Spkrs (Minimum number of first speakers)**

Min 1st Spkrs specifies the minimum number of first-speaker sessions for this mode. The valid range is 1 to the value specified for the Max Sessions setting. The default setting is 1.

### **PB Sessions (Number of prebound sessions)**

PB Sessions specifies the minimum number of first-speaker sessions that are automatically activated (prebound) when session limits are initialized. This setting corresponds to the contention winner auto-activation limit. The valid range is 1 to the value specified for the Min 1st Spkrs setting. The default setting is 1.

### **Queue Binds**

Queue Binds specifies whether binds sent across this mode can be queued if the partner is not able to accept them. Yes indicates that the function is supported and that binds can be queued. No indicates that the function is not supported, and that binds cannot be queued. The default setting is Yes.

### **Blank Mode**

Blank Mode specifies whether or not a null mode name is sent across the link. Blank mode can be provided for only one mode per partner. Yes indicates that a null mode name is sent across a link; No indicates that it is not. The default setting is No.

---

---

## **Editing defaults**

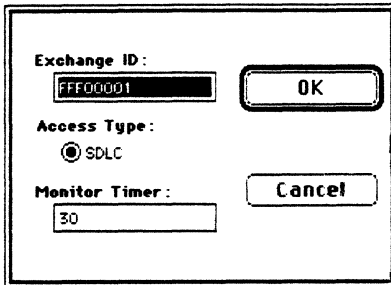
You may change any or all of the component default settings at any time. To edit defaults, choose Edit Defaults from the Edit menu. A submenu appears from which you can choose Node, Local LU, TP, Line, Partner, Remote LU, or Mode for default settings editing. When you choose one of these commands from the submenu, the appropriate dialog box with the current defaults is displayed.



---

## Node

To edit the default settings for nodes, choose Node from the Edit Defaults submenu. The dialog box in Figure 11-39 appears. See the “Local Node” section, earlier in this chapter, for an explanation of the node settings.



A dialog box titled "Node" with the following fields and controls:

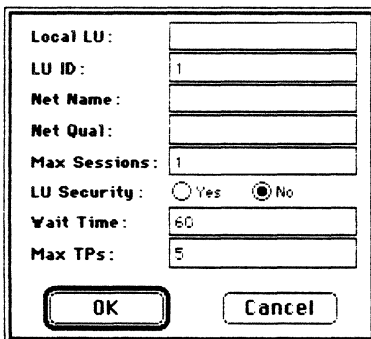
- Exchange ID:** A text field containing "FFFF0001".
- Access Type:** A radio button labeled "SDLC" is selected.
- Monitor Timer:** A text field containing "30".
- Buttons:** "OK" and "Cancel" buttons are located on the right side.

**Figure 11-39**  
Editing node default settings

---

## Local LU

To edit the default settings for local LUs, choose Local LU from the Edit Defaults submenu. The dialog box in Figure 11-40 appears. See the earlier section on editing a local LU for an explanation of the settings.



A dialog box titled "Local LU" with the following fields and controls:

- Local LU:** A text field.
- LU ID:** A text field containing "1".
- Net Name:** A text field.
- Net Qual:** A text field.
- Max Sessions:** A text field containing "1".
- LU Security:** Two radio buttons, "Yes" and "No", with "No" selected.
- Wait Time:** A text field containing "60".
- Max TPs:** A text field containing "5".
- Buttons:** "OK" and "Cancel" buttons are located at the bottom.

**Figure 11-40**  
Editing local LU default settings



---

## TP

To edit the default settings for TPs, choose TP from the Edit Defaults submenu. The dialog box in Figure 11-41 appears. See the earlier section on editing a TP for an explanation of the settings.

TP Name:

Local LU:

Net Name:

Status: ☒ Enable ☐ Temp ☐ Perm

Conv Type: ☐ Basic ☐ Map ☒ Either

Sync Level: ☒ None ☒ Confirm

PIP: ☐ Yes ☒ No

PIP Count:

PIP Check: ☐ Yes ☒ No

Data Mapping: ☐ Yes ☒ No

FMH Data: ☐ Yes ☒ No

Privilege: ☒ None ☐ CNDS ☐ Session  
☐ Define ☐ Disp ☐ Service

LUV: ☐ Yes ☒ No

Security Required:  
☒ None ☐ Conv ☐ User  
☐ Prof ☐ User/Prof

OK

Cancel

**Figure 11-41**  
Editing TP default settings

---

## Line

To edit the default settings for lines, choose Line from the Edit Defaults submenu. The dialog box in Figure 11-42 appears. See the earlier section on editing a line for an explanation of the settings.

Line Name:

Line Number: ☒ 1 ☐ 2 ☐ 3 ☐ 4

Role Type: ☐ Prim ☐ Sec ☒ Negot

Connect Type: ☒ Lease ☐ Multi ☐ Switch

Max BTU:

Line Speed: ☐ 300 ☐ 1200 ☐ 2400  
☐ 4800 ☒ 9600 ☐ 19200

Max Retries:

Idle Time:

NP Recv Time:

Max I-Frames:

NRZI Support: ☒ NRZ ☐ NRZI

Duplex Type: ☒ Half ☐ Full

OK

Cancel

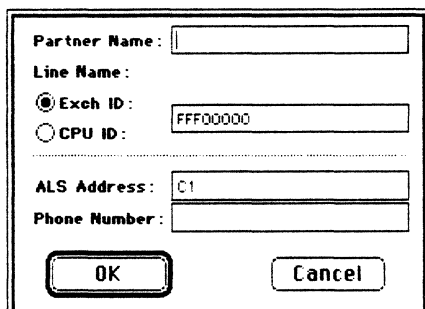
**Figure 11-42**  
Editing line default settings



---

## Partner

To edit the default settings for partners, choose Partner from the Edit Defaults submenu. The dialog box in Figure 11-43 appears. See the earlier section on editing a partner for an explanation of the settings.



A dialog box titled "Partner" with the following fields and controls:

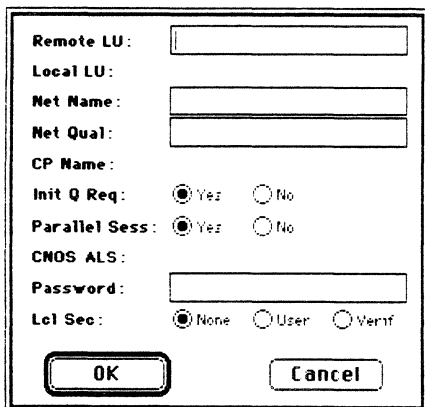
- Partner Name: [text field]
- Line Name: [text field]
- Exch ID: ☒ (selected) [text field containing "FFF00000"]
- CPU ID: ☐ [text field]
- ALS Address: [text field containing "C1"]
- Phone Number: [text field]
- OK button
- Cancel button

**Figure 11-43**  
Editing partner default settings

---

## Remote LU

To edit the default settings for remote LU, choose Remote LUs from the Edit Defaults submenu. The dialog box in Figure 11-44 appears. See the earlier section on editing a remote LU for an explanation of the settings.



A dialog box titled "Remote LU" with the following fields and controls:

- Remote LU: [text field]
- Local LU: [text field]
- Net Name: [text field]
- Net Qual: [text field]
- CP Name: [text field]
- Init Q Req: ☒ Yes ☐ No
- Parallel Sess: ☒ Yes ☐ No
- CNOS ALS: [text field]
- Password: [text field]
- Lcl Sec: ☒ None ☐ User ☐ Verif
- OK button
- Cancel button

**Figure 11-44**  
Editing remote LU default settings



---

## Mode

To edit the default settings for modes, choose Mode from the Edit Defaults submenu. The dialog box in Figure 11-45 appears. See the earlier section on editing a mode for an explanation of the settings.

Mode Name:  Sync Level: ☒ Confirm

Local LU:  Sess Reinit: ☒ None ☐ Oper ☐ Prim

Remote LU:  ☐ Sec ☐ Either

Adj Station:  Max Sessions:  1

Send Pacing:  3 Min 1st Spkrs:  1

Recv Pacing:  3 PB Sessions:  1

Max RU UB:  1024 Queue Binds: ☒ Yes ☐ No

Max RU LB:  256 Blank Mode: ☐ Yes ☒ No

OK Cancel

**Figure 11-45**  
Editing mode default settings

---

---

## Deleting network components

To delete a network component, first select the component from the appropriate list in the configuration file window (see Figure 11-28). Then choose Clear from the Edit menu.

Note that deleting components also deletes associated components:

- ☐ Deleting a local LU deletes all its TPs, remotes LUs, and modes.
- ☐ Deleting a line deletes all its partners, remote LUs, and any modes associated with the remote LUs.
- ☐ Deleting a partner deletes all its remote LUs and any modes associated with the remote LUs.
- ☐ Deleting a remote LU deletes all its modes.
- ☐ Deleting security options in a local LU causes corresponding options to be deleted in any of its TPs.

---

---

## Printing a configuration file

You can get a printout of a configuration file so that you can see all of the components and settings at once and refer to them when you are not using the configuration program.

To print a configuration file, first check Page Setup in the File menu. Then, choose Print from the File menu. An example of a configuration file printout is shown in Figure 11-46.



**Figure 11-46**

Example of a configuration file printout

11/10/88 11:25		Remote Config		Page 1	
<b>Node:</b>					
Exchange ID:	FFF00001				
Access Type:	SDLC				
Monitor Timer:	30				
<b>Local LU:</b>					
LU ID:	1				
Network Name:					
Network Qualifier:					
Max Sessions:	4				
LU Security:	No				
Wait Time:	60				
Max TPs:	10				
<b>TP Name:</b>			<b>TP Name:</b>		
Local LU:	TP1	Local LU:	TP2		
Network Name:	LU1	Network Name:	LU1		
Status:	Enabled	Status:	Enabled		
Conversation Type:	Either	Conversation Type:	Either		
Sync Level:	None, Confirm	Sync Level:	None, Confirm		
PIP Required:	No	PIP Required:	No		
PIP Count:	0	PIP Count:	0		
PIP Check:	No	PIP Check:	No		
Data Mapping:	No	Data Mapping:	No		
FMH Data:	No	FMH Data:	No		
Privilege:	None	Privilege:	None		
LUW:	No	LUW:	No		
Security Required:	None	Security Required:	None		
<b>Line Name:</b>					
Line Type:	LINE1				
Line Number:	SDLC				
Role Type:	2				
Connect Type:	Negotiable				
Max BTU:	Leased				
Line Speed:	265				
Max Retries:	9600				
Idle Time:	3				
NP Recv Time:	800				
Max I-Frames:	10000				
NRZI Support:	7				
Duplex Type:	NRZ				
	Half				
<b>Partner Name:</b>					
Line Name:	PART1				
Exchange ID:	LINE1				
ALS Address:	FFF00000				
Phone Number:	C1				

MacAPPC™ Config Version 1.0b2



**Figure 11-46**

Example of a configuration file printout (continued)

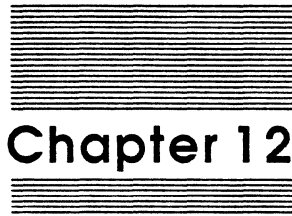
11/10/88 11:25		Remote Config		Page 2	
<b>Remote LU:</b>	LU2				
<b>Local LU:</b>	LU1				
<b>Network Name:</b>					
<b>Network Qualifier:</b>					
<b>CP Name:</b>	PART1				
<b>Init Q Req:</b>	Yes				
<b>Parallel Sessions:</b>	Supported				
<b>CNOS ALS:</b>	PART1				
<b>Password:</b>					
<b>Local Security:</b>	None				
<b>Mode Name:</b>	MODE1				
<b>Local LU:</b>	LU1				
<b>Remote LU:</b>	LU2				
<b>Adjacent Station:</b>	PART1				
<b>Send Pacing:</b>	3				
<b>Receive Pacing:</b>	3				
<b>Max RU UB:</b>	1024				
<b>Max RU LB:</b>	256				
<b>Sync Level:</b>	Confirm				
<b>Session Reinitiation:</b>	Not Supported				
<b>Max Sessions:</b>	4				
<b>Min 1st Spkrs:</b>	2				
<b>PB Sessions:</b>	1				
<b>Queue Binds:</b>	Yes				
<b>Blank Mode:</b>	No				

MacAPPC™ Config Version 1.0b2









## **Chapter 12**

# **The MacAPPC Administration Program**



This chapter describes the Administration program and how it is used to monitor and control a MacAPPC network.

The Administration program simplifies the task of network implementation by providing a clear, organized Macintosh user interface to assist in the dynamic real-time management of the network.

The Administration program provides a set of functions for starting and stopping the MacAPPC server, and for displaying, activating, and deactivating network components and sessions. Specifically, it can

- start a MacAPPC server on an intelligent communications card, configuring it using a file created by the Configuration program
- stop a MacAPPC server
- display network components (local node, lines, stations, control points, local LUs, transaction programs, remote LUs, modes) and sessions
- activate and deactivate individual network components (lines, stations, local LUs) and sessions
- initialize and reset session limits for individual modes
- log internal MacAPPC server log messages to a file according to specified criteria

---

---

## The Administration program menu bar

The Administration program has four menus in addition to the Apple menu: File, Edit, Server, and Log. The File menu, illustrated in Figure 12-1, permits you to close the active window or quit the Administration program.

The Edit menu, illustrated in Figure 12-2, has the standard Macintosh user interface commands of Undo, Cut, Copy, Paste, and Clear for editing.

File	
Close	⌘W
<hr/>	
Quit	⌘Q

**Figure 12-1**  
The File menu

Edit	
Undo	⌘Z
<hr/>	
Cut	⌘H
Copy	⌘C
Paste	⌘V
Clear	

**Figure 12-2**  
The Edit menu

The Server menu, illustrated in Figure 12-3, has MacAPPC commands that provide convenient and efficient interaction and control of the servers located on the intelligent communication cards. The commands in the Server menu are explained later in this chapter.



The Log menu, illustrated in Figure 12-4, provides commands that relate to the log, a detailed chronological record of network activity. The log may be turned on, turned off, erased, updated periodically or displayed in real time, and controlled to provide the level of information detail required. See the section on logging later in this chapter for explanations of the Log menu commands.

Server	
Start Server...	
Stop Server...	
Display Update	%U
Activate	%A
Deactivate	%D
Start CNOS	
Stop CNOS	

**Figure 12-3**  
The Server menu

Log	
Active	%L
Settings...	
Show Log	
Clear Log...	

**Figure 12-4**  
The Log menu

---

---

## Conventions used in the Administration program

This section describes some conventions used in the Administration program that you should know about before proceeding.

---

### Special cursor

A “spinning beach ball” cursor appears in the screen whenever the Administration program is talking to the Server. The beach ball cursor spins during the brief periods of interaction and then disappears when the required information has been transferred.

---

### Network display control

A network display control is used in the server window to display any of four levels of the network components and sessions summary. This feature is especially useful for displaying the variable levels of detail of a complex network configuration. The network display control is described in the section on displaying network components and sessions.

---

### Severity control

A severity control is used to select the detail level of logging messages. This control may be adjusted from messages of low to high detail. The severity control is described in the section on log settings options.



---

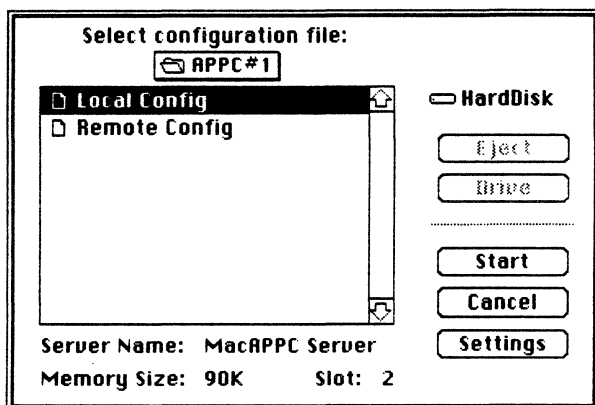
---

## Starting a MacAPPC server

Before starting a server, be sure that one or more configuration files have already been created, and that an intelligent network communications card has been properly installed in the Macintosh II. See Chapter 9, "Installation," and Chapter 11, "The MacAPPC Configuration Program," for further information.

To start a MacAPPC server, choose Start Server from the Server menu. The dialog box shown in Figure 12-5 appears. This dialog box is similar to the standard directory dialog box, which allows you to move around in the hierarchical file system and select the file you want to work with. The current list of configuration files is displayed. The dialog box also displays the current server name, the amount of memory on the communications card that is reserved for the MacAPPC server, and the NuBus slot number (1, 2, 3, 4, 5, or 6) where the server resides.

Select the desired configuration file and click Start to start the server. The watch appears while the server is being downloaded. The spinning beach ball appears when the server is being configured. The screen then returns to the Administration program menu bar. If you see the message "The Macintosh has no available communication cards," it means that no communication cards have been installed.



**Figure 12-5**  
Selecting a configuration file

If you want to change the name, memory size, or slot number, click Settings before starting the server. A dialog box similar to the one in Figure 12-6 appears. A dimmed slot number (1, 2, 3, 4, 5, or 6) indicates that there is no intelligent communications card in that slot; an active slot number indicates that there is a communications card in that slot. A server name appearing to the right of the slot indicates the server that has been started and is currently running on the communications card in the slot. This example shows that both slots 2 and 4 have communication cards installed. Slot A has a server named "MacAPPC Server" running on it. If the current settings are used, a server named "Slot 4 Server" with a memory size of 90K would be started on the communications card in slot 4.



Server Name: Slot 4 Server	
Memory Size: 90 K	
Slot	Server Name
<input type="radio"/> 1	
<input type="radio"/> 2	MacAPPC Server
<input type="radio"/> 3	
<input checked="" type="radio"/> 4	
<input type="radio"/> 5	
<input type="radio"/> 6	
<input type="button" value="Cancel"/> <input type="button" value="OK"/>	

**Figure 12-6**  
Editing server settings

### Server Name

Server Name specifies the server name that appears in the Chooser. See Chapter 10, "Selecting a MacAPPC Server," for additional information. The default name is MacAPPC Server.

### Memory Size

Memory Size specifies the amount of memory to be reserved on the intelligent communications card for the MacAPPC server. The default setting is 90K.

### Slot

Slot specifies the NuBus slot number for the intelligent communications card where the currently active server resides. The default setting is slot 2.

---

---

## Displaying network components and sessions

The Administration program permits you to display network components and sessions and their associated settings. First select a MacAPPC server using the Chooser. Then, choose Display from the Server menu. The server window in Figure 12-7 appears. Information about the local node is displayed in the upper-left corner. The lists display component names that are defined for the current network configuration and their respective session IDs. Note that unlike the Configuration program, all network components (except for TPs) are displayed at once. Transaction programs are listed in a specific local LU window described later in this chapter.



**MacAPPC Server**

**Exchange ID:** FFF00001

**Access Type:** ☒ SDLC

**Monitor Timer:** 50

Lines		Stations	
•LINE1		PART1	

Local LUs	Remote LUs	Modes	Sessions
•LU1	LU2	•MODE1	38824
•LU2	LU1	•MODE1	38220

CNOS service TPs in process locally : 0

**Figure 12-7**  
The server window

### Exchange ID

Exchange ID contains the identifier of the remote node if the node is a peer. It is used in XID exchange at link establishment.

### Access Type

Access Type specifies the type of line. Synchronous Data Link Control (SDLC) is the only access type currently available.

### Monitor Timer

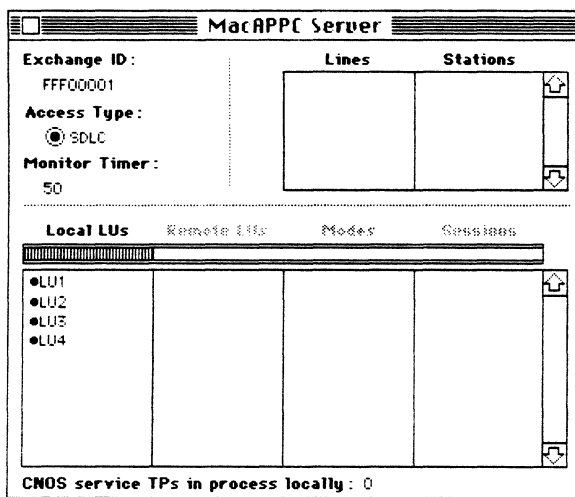
Monitor Timer specifies the wake-up interval for the program monitor in seconds. The wake-up interval is the time the server waits for a reply from an attached TP. If the interval time is exceeded, the server disconnects.

Lines, stations, local LUs, and sessions can be activated or deactivated from the server window. Session limits can be initialized and reset by activating or deactivating modes. A dot “•” appears to the left of an activated component and disappears when a component is deactivated. The number of CNOS TPs in process locally is displayed. These settings, as well as activation and deactivation of components and sessions, are described later in this chapter in the section “Managing Network Components and Sessions.”



All of the local LUs, remote LUs, modes, and sessions are displayed simultaneously for a simple network configuration, such as the example shown in Figure 12-7. For a more complex configuration that cannot be displayed at once, the lists may be moved up or down with the scroll bar. Alternatively, you may use the network display control feature located between the titles Local LUs, Remote LUs, Modes, and Sessions and their respective lists to display any of four levels of the outlined component and sessions summary. To use the network display control, position the cursor in the right end of the shaded area. The cursor becomes a double arrow. Then, drag the cursor to display the level of components and sessions desired. The information displayed is at a maximum when the cursor is moved to the right end, the Sessions level, as it is in Figure 12-7.

A more complicated configuration example will demonstrate the usefulness of the variable levels of detail provided by the network display control. The example in Figure 12-8 shows a network configuration with the control set at the local LU level of information. The four local LUs of the network configuration are displayed.



**Figure 12-8**  
Displaying components at the local LU level



Figure 12-9 shows the lists of the example configuration at the remote LU level. Both local LUs and remote LUs are displayed.

MacAPPC Server

Exchange ID:  
FFF00001

Access Type:  
☒ SDLC

Monitor Timer:  
50

Lines

Stations

Local LUs

Remote LUs

Modes

Sessions

●LU1

●LU2

●LU3

●LU4

LU2

LU3

LU1

LU4

LU1

LU2

CNOS service TPs in process locally : 1

Figure 12-9  
Displaying components at the remote LU level

Figure 12-10 shows the lists of the example configuration at the modes level. Local LUs, remote LUs, and modes that are hidden may be brought into view using the scroll bar.

MacAPPC Server

Exchange ID:  
FFF00001

Access Type:  
☒ SDLC

Monitor Timer:  
50

Lines

Stations

Local LUs

Remote LUs

Modes

Sessions

●LU1

●LU2

●LU3

LU2

LU3

LU1

LU4

LU1

●SNASVCMG  
MODE12

●SNASVCMG  
MODE13

●SNASVCMG  
MODE12

●SNASVCMG  
MODE24

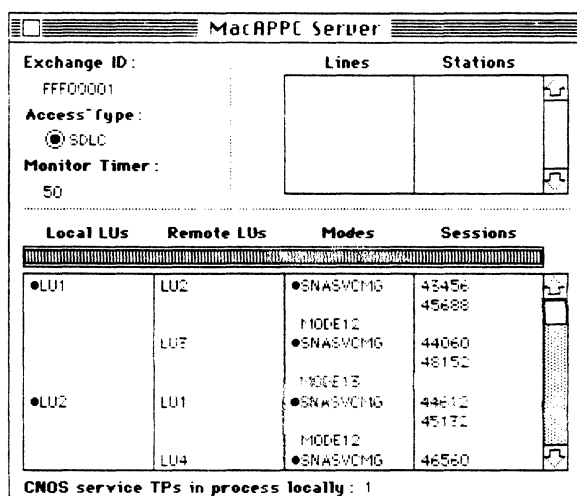
●SNASVCMG  
MODE13

CNOS service TPs in process locally : 1

Figure 12-10  
Displaying components at the mode level



Figure 12-11 shows the lists at the sessions level. Local LUs, remote LUs, modes, and sessions that are hidden from view may be brought into view using the scroll bar.



**Figure 12-11**  
Displaying components and sessions at the session level

You may select components or sessions from the lists to display their settings. Double-click a line, station, local LU, remote LU, mode, or session. The appropriate window appears displaying the settings defined by the current configuration. To display transaction program settings, first double-click a local LU. A local LU window appears displaying a list of TPs. Double-click the TP of interest. A TP-settings display window appears. See the sections “Displaying a Local LU” and “Displaying a Transaction Program,” later in this chapter, for additional details.

Most of the settings displayed are for review only. However, dynamic settings in the local LU, line, station and control point, mode, and session display windows change as components and sessions are activated and deactivated and conversations are allocated and deallocated.

## Displaying a local LU

To display the settings for a local LU, double-click the local LU in the Local LUs list in the server window (see Figure 12-7). The local LU display window in Figure 12-12 appears. The local LU settings displayed are for review only. The only exception is Act Sess. Act Sess is a dynamic setting that changes when sessions are activated or deactivated for this local LU. To make changes in local LU settings, you must use the Configuration program to create components and edit their settings (see Chapter 11).



Local LU: LU1			
Local LU:	LU1	Max Sess:	10
LU ID:	1	Act Sess:	0
Net Name:	LU1	Wait Time:	60
Net Qual:		Max TPs:	10
LU Sec:	<input type="radio"/> Yes <input checked="" type="radio"/> No		
User IDs		Profiles	Transaction Programs
			CNOS ADMIN TP1 TP2

**Figure 12-12**  
A local LU display window

## Local LU

Local LU is the name used to associate mode, remote LU, and transaction program definitions with a particular local LU.

## LU ID

LU ID is the local LU ID number. The setting corresponds to the destination address field (DAF).

## Net Name (Local LU network name)

Net Name is the name by which the local LU is known throughout the network.

## Net Qual (Local LU network qualifier)

Net Qual displays the current value for the local LU network qualifier.

## LU Sec (LU security)

LU Sec indicates whether or not conversation-level security is enabled for this local LU. If it is, the user ID and the password are checked at the conversation level, regardless of the TP-defined security. Yes specifies that conversation-level security is enabled for the local LU; No specifies that it is not.

## Max Sess (Maximum number of sessions)

Max Sess is the maximum number of sessions that can be active at the local LU at one time. Session limits are currently maintained only at a mode level.

## Act Sess (Number of active sessions)

Act Sess specifies the number of sessions currently active for this local LU.

## Wait Time

Wait Time specifies the amount of time in seconds the LU waits for routine completion.



## Max TPs (Maximum transaction programs)

Max TPs specifies the maximum number of remote or local transaction programs that can be attached to the local LU at one time.

## User IDs

User IDs are network security settings. A user ID is required only when the security permits user IDs on the resource-access authorization list. Double-click a user ID name to display its security password. User IDs that are not visible can be brought into view using the scroll bar. Selecting a user ID has no effect.

## Profiles

Profiles are network security settings. A profile is required only when the security permits profiles on the resource-access authorization list. Each user ID is displayed with its associated profiles. User IDs and profiles that are not visible can be brought into view using the scroll bar. Selecting a profile has no effect.

## Transaction Programs

Transaction Programs displays a list of all of the transaction programs defined for this local LU. TPs that are not visible can be brought into view using the scroll bar.

---

## Displaying a transaction program

To display the settings for a transaction program, double-click the TP displayed in the Transaction Programs list in the local LU display window. The window in Figure 12-13 appears. The TP settings displayed are for review only. To change TP settings, you need to create components and edit their settings using the Configuration program (see Chapter 11).

The screenshot shows a window titled "TP: TP 1". It contains the following settings:

- TP Name:** TP1
- Local LU:** LU1
- Net Name:**
- Status:** ☒ Enable ☐ Temp ☐ Perm
- Conv Type:** ☐ Basic ☐ Map ☒ Either
- Sync Level:** ☒ None ☒ Confirm
- PIP:** ☐ Yes ☒ No
- PIP Count:** 0
- PIP Check:** ☐ Yes ☒ No
- Data Mapping:** ☐ Yes ☒ No
- FMH Data:** ☐ Yes ☒ No
- LUV:** ☐ Yes ☒ No
- Privilege:** ☒ None ☐ CNOS ☐ Session ☐ Define ☐ Display ☐ Service
- Security Required:** ☒ None ☐ Conv ☐ User ☐ Profile ☐ User/Prof
- User IDs:** (Empty list box)
- Profiles:** (Empty list box)

**Figure 12-13**

A transaction program display window



## **TP Name**

TP Name specifies the name of this TP; it is unique for this local LU. The names *CNOS* and *ADMIN* are reserved by MacAPPC.

## **Local LU**

Local LU specifies the name of the LU with which the TP is associated.

## **Net Name (Transaction program network name)**

Net Name is the name by which the TP is known throughout the network.

## **Status**

Status specifies the response the LU makes when an allocation request is received that designates this TP. The program may be enabled, temporarily removed, or permanently removed from network availability.

**Enable (Enabled):** The TP is enabled.

**Temp (Temporary):** The TP is temporarily disabled.

**Perm (Permanent):** The TP is permanently disabled.

## **Conv Type (Conversation type)**

Conv Type specifies the conversation type allowed on an allocation request to start the program.

**Basic (Basic conversation type):** The TP may be allocated only with a basic conversation.

**Map (Mapped conversation type):** The TP may be allocated only with a mapped conversation.

**Either (Either conversation type):** The TP may be allocated with either a basic or a mapped conversation.

## **Sync Level (Synchronization level)**

Sync Level specifies the synchronization level allowed on an allocation request to start the program.

**None:** The synchronization level is none.

**Confirm:** The synchronization level is confirm.

## **PIP (Program initialization parameters)**

PIP indicates whether or not this transaction program supports program initialization parameters. Yes indicates that PIP is supported; No indicates that it is not.

## **PIP Count**

PIP Count is the program initialization parameters count. It specifies the number of PIP subfields that may be required in order to start this transaction program.



## PIP Check

PIP Check specifies whether or not this transaction program supports program initialization parameter checking. No indicates that an allocation request for the TP is not rejected, even if the number of PIP subfields in the PIP Count do not match the number supplied on the request. Yes indicates that an allocation request for the transaction program is rejected if the number of PIP subfields in the PIP Count do not match the number supplied on the request.

## Data Mapping

Data Mapping specifies whether or not this transaction program is provided with data-mapping support. Yes indicates that data mapping is supported; No indicates that it is not.

## FMH Data

FMH Data indicates whether or not the functional management header (FMH) data can be transmitted and received by this transaction program. Yes indicates that the transaction program can transmit and receive this information; No indicates that it cannot.

## LUW (Logical unit of work)

LUW indicates whether or not this transaction program assigns logical-unit-of-work IDs for each transaction issued by this program. Yes specifies that the TP assigns LUW IDs; No specifies that it does not.

## Privilege

**None:** No routines that require a privilege are allowed.

**CNOS:** CNOS routines are allowed.

**Session:** Session-control routines are allowed.

**Define:** Definition routines are allowed.

**Display:** Display routines are allowed.

**Service:** Allocation routines may be issued. The TP name designates a service transaction program, such as SNADS or DIA.

## Security Required

Security Required specifies the security verification provided on an allocation request to start the program.

**None:** No security checking is performed for this TP. Note that if LU security is set to Yes for the local LU, security is checked at the local LU level, even if Security Required is None.

**Conv (Conversation-level security):** Conversation-level security is specified. If security parameters are specified on a conversation allocation, they are checked at the LU level only.

**Profile:** Resource-access security is required for this TP. The profile must match a profile defined on the resource-access authorization list.



**User (User ID):** Resource-access security is required. The user ID must match a user ID defined on the resource-access authorization list.

**User/Prof (User ID and profile):** Resource-access security is required. The user ID and profile must match both a user ID and a profile on the resource-access authorization list.

## User IDs

User IDs are network security settings. A user ID is required only when the security permits user IDs on the resource-access authorization list. User IDs that are not visible can be brought into view using the scroll bar. Selecting a user ID has no effect.

## Profiles

Profiles are network security settings. A profile is required only when the security permits profiles on the resource-access authorization list. Each user ID is displayed with its associated profiles. User IDs and profiles that are not visible can be brought into view using the scroll bar. Selecting a profile has no effect.

---

## Displaying a line

To display the settings for a line, double-click the line displayed in the Lines list in the server window (see Figure 12-7). The window in Figure 12-14 appears. The line settings displayed are for review only. The only exception is Line Status. Line Status is a dynamic setting that may change when a line is activated. To change line settings, you need to create components and edit their settings using the Configuration program (see Chapter 11).

**Line: LINE1**

**Line Name:** LINE1  
**Line Type:** SDLC  
**Line Status:** ☐ Reset ☐ PendActive  
☒ Active ☐ PendReset

---

**Line Number:** ☒ 1 ☐ 2 ☐ 3 ☐ 4  
**Role Type:** ☐ Prim ☐ Sec ☒ Negot  
**Connect Type:** ☒ Lease ☐ Multi ☐ Switch  
**Max BTU:** 265  
**Line Speed:** ☐ 300 ☐ 1200 ☐ 2400  
☐ 4800 ☒ 9600 ☐ 19200  
**Max Retries:** 3  
**Idle Time:** 800  
**NP Recv Time:** 10000  
**Max I-Frames:** 7  
**NRZI Support:** ☒ NRZ ☐ NRZI  
**Duplex Type:** ☒ Half ☐ Full

**Figure 12-14**  
A line display window

## Line Name

Line Name is the name by which the line is known throughout the network.



## Line Type

Line Type is based on the access type of the local node. Synchronous Data Link Control (SDLC) is the only line type that is currently available.

## Line Status

**Reset:** The line status is reset.

**PendActive (Pending active):** The line status is active pending a response.

**Active:** The line status is active.

**PendReset (Pending reset):** The line status is pending reset.

## Line Number

Line Number specifies the serial port number to which the line is connected. The line number may be 1, 2, 3, or 4.

## Role Type

Role Type specifies whether or not the Synchronous Data Link Control (SDLC) role is primary, secondary, or negotiable.

**Prim (Primary):** The SDLC role is primary.

**Sec (Secondary):** The SDLC role is secondary.

**Negot (Negotiable):** The SDLC role is negotiable.

## Connect Type

**Lease (Leased):** The connection is leased.

**Multi (Multipoint):** The connection is multipoint.

**Switch (Switched):** The connection is switched.

## Max BTU (Maximum basic transmission unit length)

Max BTU specifies the maximum basic transmission unit length that can be received on this line. This value is exchanged with the partner at link initiation.

## Line Speed

Line Speed specifies the transmission line speed in bits per second (bps).

## Max Retries (Maximum number of retries)

Max Retries specifies the maximum number of times a frame is retransmitted after SDLC procedures have detected a discrepancy. When this number is exceeded, SDLC reports the problem to a higher level of SNA for resolution.



## Idle Time

Idle Time specifies the amount of time in milliseconds that can elapse without a response before a primary link station initiates recovery action.

## NP Recv Time (Nonproductive receive time)

NP Recv Time specifies the amount of time that can elapse before a primary link station reports to a higher level of SNA that a nonproductive receive exists for resolution.

## Max I-Frames (Maximum number of I-frames)

Max I-Frames specifies the maximum number of I-frames that can be sent before polling resumes.

## NRZI Support

NRZI Support specifies the type of transmission encoding method to be used when sending signals over this line.

**NRZ:** A nonreturn-to-zero (NRZ) encoding method is used.

**NRZI:** A nonreturn-to-zero-inverted (NRZI) encoding method is used.

## Duplex Type

Duplex Type specifies whether data can be sent in one direction or both directions without turning the line around.

**Half:** The transmissions are half-duplex. Data can be sent in only one direction and then the line must be turned around.

**Full:** The transmissions are full-duplex. Data can be sent and received without turning the line around.

---

## Displaying the station and control point

To display the settings for a station and control point, double-click the station displayed in the Stations list in the server window (see Figure 12-7). The window in Figure 12-15 appears. The station and control point settings displayed are for review only. The only exception is Status. Status is a dynamic setting that may change when the station is activated. To change other station settings, you need to create components and edit their settings using the Configuration program (see Chapter 11).

The Administration program assumes that a station is defined for every control point. This is the case if you configure the server using a configuration file, since a partner in the Configuration program consists of both a station and control point. If you write a program to define your own control point without defining a station for it, its information is not displayed. Note that if you start the server using a configuration file, the control point name (CP Name) is always the same as the station name, that is, the partner name in the Configuration program.



Station: PART1	
Station Name:	PART1
Line Name:	LINE1
Status:	<input checked="" type="radio"/> Reset <input type="radio"/> PendResp <input type="radio"/> PendCont <input type="radio"/> Active <input type="radio"/> PendReset <input type="radio"/> RstPenResp
ALS Address:	C1
Phone Number:	
<hr/>	
CP Name:	PART1
<input checked="" type="radio"/> Exch ID:	FFF00000
<input type="radio"/> CPU ID:	

**Figure 12-15**  
A station display window

### Station Name

Station Name specifies the name by which the station is known throughout the network. This name is the same as CP Name if the server was configured with a configuration file.

### Line Name

Line Name specifies the line name associated with this station.

### Status

**Reset:** The station status is reset.

**PendResp (Pending response):** The station status is active pending response.

**PendCont (Pending contact):** The station status is active pending contact.

**Active:** The station status is active.

**PendReset (Pending reset):** The station status is pending reset.

**RstPenResp (Reset pending response):** The station status is reset pending response.

### ALS Address (Adjacent-link-station address)

ALS Address is the adjacent-link-station address.

### Phone Number

Phone Number specifies the phone number or dialing code for this station.

### CP Name (Control point name)

CP Name specifies the name of the control point. This name is the same as the station name if the server was configured with a configuration file.

### Exch ID (Exchange ID) or CPU ID

Either the exchange ID or the CPU ID is specified.

### Exch ID

Exch ID specifies the exchange ID used in XID exchange at link establishment.



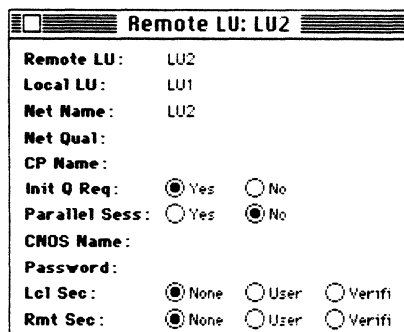
## CPU ID

CPU ID specifies the CPU identifier.

---

## Displaying a remote LU

To display the settings for a remote LU, double-click the remote LU displayed in the Remote LUs list in the server window (see Figure 12-7). The window in Figure 12-16 appears. The settings displayed are for review only. The only exception is Rmt Sec. Rmt Sec is a dynamic setting that changes after a session is bound. To change remote LU settings, you need to create components and edit their settings using the Configuration program (see Chapter 11).



**Figure 12-16**  
A remote LU display window

### Remote LU

Remote LU specifies the name by which the local LU can recognize the remote LU.

### Local LU

Local LU specifies the name of the local LU for which the remote LU has been defined.

### Net Name (Remote LU network name)

Net Name specifies the name by which the remote LU is known throughout the network.

### Net Qual (Remote LU network qualifier)

Net Qual is an optional field that may be defined for cross-network communications to make the network name unique.

### CP Name (Control point name)

CP Name is the name of the control point where the remote LU is located. In a local configuration, this field is blank.



### **Init Q Req (Queue session-initiation requests)**

Init Q Req specifies whether or not system services control point (SSCP) should queue session-initiation requests if the remote LU is not able to accept them. Yes indicates that SSCP queues session-initiation requests if the remote LU is not able to accept them; No indicates that it does not.

### **Parallel Sess (Parallel sessions)**

Parallel Sess specifies whether or not the remote LU supports parallel sessions. This information is used during session activation by BIND requests and responses. Yes indicates that this function is supported; No indicates that it is not. Note that if parallel sessions are supported, you must have a CNOS transaction program running to initialize user modes.

### **CNOS Name**

CNOS Name specifies the name of the station that handles change-number-of-sessions (CNOS) requests. If the local LU and remote LU are in the same node, this name is *LOCAL*. Otherwise, it's the name of a previously defined station control block. If parallel sessions are supported, a name must be specified. If parallel sessions are not supported, this field is blank. In general, the CNOS name is the partner name.

### **Password**

Password specifies the password defined for the remote LU. It is used to provide session-level verification during session activation.

### **Lcl Sec (Local security)**

Lcl Sec specifies the access-security information that the local LU accepts for the remote LU.

**None:** No security is accepted.

**User:** Access-security information is accepted. The already verified indication is not accepted.

**Verifi (Verified):** Access-security information is accepted. The already verified indication is accepted.

### **Rmt Sec (Remote security)**

Rmt Sec is a dynamic setting that specifies the remote LU's view of the local security. This is determined after a session is bound. The remote security setting displayed is not valid before the session is bound.

**None:** No security is accepted.

**User:** Access-security information is accepted. The already verified indication is not accepted.

**Verifi (Verified):** Access-security information is accepted. The already verified indication is accepted.



## Displaying a mode

To display the settings for a mode, double-click the mode displayed on the Modes list in the server window (see Figure 12-7). The window in Figure 12-17 appears. Most of the mode settings displayed are for review only. The exceptions are the current and active Max Sessions, Min 1st Spkrs, and Min Bidders. These dynamic settings change as sessions are activated and deactivated and modes are initialized and reset. To change other mode settings, you need to create components and edit their settings using the Configuration program (see Chapter 11).

Mode: MODE1			
Mode Name:	MODE1		
Local LU:	LU1	Send Pacing:	3
Remote LU:	LU2	Recv Pacing:	3
Adj Station:	LOCAL	Max RU UB:	1024
Sync Level:	<input checked="" type="radio"/> Confirm	Max RU LB:	256
PB Sessions:	1	Term Count:	0
		Sess Reinit:	<input checked="" type="radio"/> Oper <input type="radio"/> Prim
			<input type="radio"/> Sec <input type="radio"/> Either
		Queue Binds:	<input checked="" type="radio"/> Yes <input type="radio"/> No
Max Sessions:	Def: 4, Curr: 4, Act: 1	Blank Mode:	<input type="radio"/> Yes <input checked="" type="radio"/> No
Min 1st Spkrs:	2, 2, 1	Drain Local:	<input type="radio"/> Yes <input checked="" type="radio"/> No
Min Bidders:	2, 2, 0	Drain Remote:	<input type="radio"/> Yes <input checked="" type="radio"/> No

**Figure 12-17**  
A mode display window

### Mode Name

Mode Name specifies the name of the group of sessions that will have the characteristics defined by the mode's settings. *SNASVCMG* is a reserved mode name.

### Local LU

Local LU specifies the local LU name for which the mode is defined.

### Remote LU

Remote LU specifies the remote LU name for which the mode is defined.

### Adj Station (Adjacent station name)

Adj Station specifies the adjacent station name. *LOCAL* indicates that the local LU and the remote LU are in the same node.

### Sync Level (Synchronization level)

Sync Level specifies the synchronization level allowed on an allocation request to start a transaction program. Confirm is the only synchronization level currently supported.

### PB Sessions (Number of prebound sessions)

PB Sessions specifies the minimum number of PB first-speaker sessions that are automatically activated when session limits are initialized.



**Max Sessions (Maximum number of sessions)**

**Def (Defined):** The maximum number of sessions that have been defined for this network configuration.

**Curr (Current):** The number of currently available sessions.

**Act (Active):** The number of active sessions.

**Min 1st Spkrs (Minimum number of first speakers)**

**Def (Defined):** The minimum number of first speaker sessions that have been defined for this network configuration.

**Curr (Current):** The number of currently available first speakers sessions.

**Act (Active):** The number of active first speaker sessions.

**Min Bidders (Minimum number of bidders)**

**Def (Defined):** The minimum number of bidders that have been defined for this network configuration.

**Curr (Current):** The number of currently available bidder sessions.

**Act (Active):** The number of active bidder sessions.

**Send Pacing**

Send Pacing specifies the number of requests the local LU can send before receiving a pacing response. This is used to determine the send window size during session activation.

**Recv Pacing (Receive pacing)**

Recv Pacing specifies the number of requests the local LU can receive. This is used to determine the receive window size during session activation.

**Max RU UB (Maximum request/response unit upper bound)**

Max RU UB specifies the upper bound on the maximum request/response unit (RU) size that can be sent or received across this mode. This setting is used in conjunction with the maximum RU lower-bound setting to determine the maximum RU size possible during session activation.

**Max RU LB (Maximum request/response unit lower bound)**

Max RU LB specifies the lower bound of the maximum request/response unit (RU) size that can be sent or received across this mode. This field is used in conjunction with the maximum RU upper-bound setting to determine the maximum RU size possible during session activation.

**Term Count (Termination count)**

Term Count specifies the number of sessions the local LU is responsible for deactivating as a result of CNOS negotiations.



## Session Reinit (Session reinitiation)

Session Reinit specifies the responsibility for a single-session reinitiation.

**Oper (Operator reinitiation):** An operator from either the local or the remote LU attempts session reinitiation. Neither LU attempts automatic reinitiation.

**Prim (Primary LU):** The primary LU automatically attempts the reinitiation.

**Sec (Secondary LU):** The secondary LU automatically attempts the reinitiation.

**Either (Either primary or secondary):** Either the primary or the secondary LU automatically attempts the reinitiation.

## Queue Binds

Queue Binds specifies whether binds sent across this mode can be queued if the partner is not able to accept them. Yes indicates that the function is supported and that binds can be queued. No indicates that the function is not supported, and that binds cannot be queued.

## Blank Mode

Blank Mode specifies whether or not a null mode name is sent across the link. Blank mode can be provided for only one mode per partner. Yes indicates that a null mode name is sent across a link; No indicates that it is not.

## Drain Local

Drain Local indicates whether or not the local LU is allowed to drain allocation requests. Yes indicates that it may; No indicates that it may not.

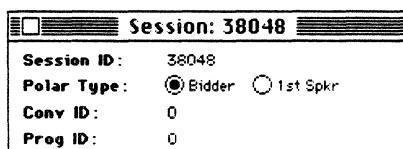
## Drain Remote

Drain Remote indicates whether or not the remote LU is allowed to drain allocation requests. Yes indicates that it may; No indicates that it may not.

---

## Displaying a session

To display the settings for a session, double-click the session displayed in the Sessions list in the server window (see Figure 12-7). The window in Figure 12-18 appears. Session ID and Polar Type are defined when the Session is activated and are for review only. Conv ID and Prog ID are dynamic settings that change when a conversation is allocated or deallocated.



**Figure 12-18**

A session display window



## Session ID

Session ID specifies the identifier number of the LU-LU session. This number is assigned by the Administration program. Each session ID is unique in the network.

## Polar Type (Polarity type)

**Bidder:** The half-session is a bidder, as determined during the bind, and may not initiate a bracket without requesting permission from the first speaker.

**1st Spkr (First speaker):** The half-session is a first speaker, as determined during the bind, and may initiate a bracket.

## Conv ID (Conversation ID)

Conv ID specifies the conversation identifier that is using this session. If the session is not being used by a conversation, this value is 0.

## Prog ID (Program ID)

Prog ID specifies the transaction program ID of the program that is using this session. If the session is free, this value is 0.

---

---

# Managing network components and sessions

MacAPPC makes managing the network simple by providing a convenient user interface to assist you in updating the server window, starting and stopping CNOS, and activating and deactivating components and sessions.

---

## Updating the server window

To update the most current information for all network components and sessions, choose Update from the Server menu. The spinning beach ball appears while the server is polled and disappears when the current information is displayed. The Administration program displays the most current information for all network components and sessions.

---

## Important

The state of the MacAPPC server can change without the Administration program being aware of it. For example, transaction programs can allocate conversations, or a remote node can activate a session. In general, it is a good idea to update the server window before activating or deactivating components.

---



---

## Starting and stopping CNOS

CNOS (change number of sessions) is the SNA service transaction program that negotiates session limits over modes between LUs that support parallel sessions. Essentially, it is a TP that is allocated and executes one routine, `COProcessSessionLimit`.

CNOS is invoked whenever a remote node (or local node in the intranode case) attempts to initialize, change, or reset session limits on user modes between LUs that support parallel sessions. These three routines are known as CNOS routines. Note that if the remote node never issues any of these routines (that is, all CNOS routines are issued locally), CNOS does not need to be present on the local node. The Administration program considers activating a mode the same as initializing session limits to their defined maximums. Deactivating a mode is equivalent to resetting session limits. The Administration program does not issue the change session limits routine.

The CNOS TP provided with MacAPPC is started and stopped from within the Administration program, but runs independently from applications. Thus, you can quit the Administration program while one or more CNOS TPs are still running. A CNOS TP can only be started from the Macintosh II that the server resides in. When a server is stopped, the Administration program first stops all CNOS TPs for the server. If more than one server is on a Macintosh II, CNOS TPs can be started for each.

To start a CNOS TP, first display a MacAPPC server from the Macintosh II it resides in. Then choose Start CNOS from the Server menu. The count of CNOS TPs in process locally is displayed at the bottom of the server window. The count is incremented by one when a CNOS TP is started. To stop a CNOS TP, choose Stop CNOS from the Server menu. The count of CNOS TPs in process is decremented by one when a CNOS TP is stopped. If a CNOS TP is currently processing a CNOS routine, the Administration program waits for it to complete the processing before stopping.

There is a small delay after CNOS has finished processing before it is reattached to the server. During this time it is possible for another CNOS routine to arrive for processing and cause a failure because CNOS is not available. Thus, if you expect frequent requests to modify session limits, you may wish to start multiple instances of the CNOS TP so that one is always available.

CNOS is allocated over the SNASVCMG mode. This mode is defined for all remote LUs that support parallel sessions. Initializing the SNASVCMG mode does not require CNOS to be present since the session limits for this mode are defined by SNA. This mode must be initialized before any user modes because it is needed by CNOS to communicate.

In summary, you need to start one or more CNOS TPs when at least one LU-LU pair has the SNASVCMG mode defined (implying parallel sessions are supported). Either the remote node issues CNOS routines or the configuration is intranode, that is, both LUs reside within the MacAPPC server.



---

## Activating network components and sessions

Once a MacAPPC server has been started, network components and sessions need to be activated from the server window before actual communication can take place. Activated network components are prefixed with a dot “•” in the server window. Active sessions are indicated by the existence of a session number. An activated mode is a mode that has had session limits initialized, that is, the current number of sessions is greater than zero. Mode activity may be monitored by updating the mode display and noting the number of active sessions, first speakers, and bidders.

### Order of activation

In general, the order of component and session activation should be as follows:

1. Lines
2. Stations (if multipoint or switched)
3. Local LUs
4. SNASVCMG modes
5. User modes
6. Sessions

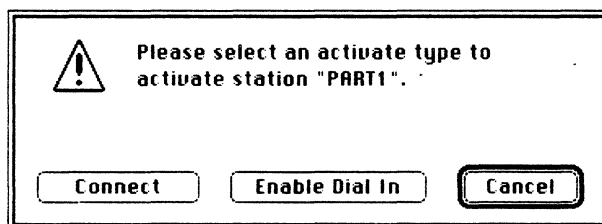
### Lines

To activate a line, select the line and choose Activate from the Server menu. To activate all of the lines on the list, click the title of the list (Lines) and choose Activate from the Server menu. Activate the line before activating the station.

### Stations

To activate a station, select the station and choose Activate from the Server menu. To activate or deactivate all of the stations on the list, click the title of the list (Stations) and choose Activate from the Server menu. The dialog box in Figure 12-19 appears to prompt you for the activate type.

You need to activate the station only after activating a switched or multipoint line. With a dedicated (leased) line, the station is activated when contact with the remote node is made.



**Figure 12-19**  
Activating a station



**Connect:** For a multipoint line, an attempt will be made to connect, and for a switched line, an attempt will be made to dial out. If a phone number has been defined for the station and logging is turned on, a message indicating that the number should be dialed will be displayed in the log window.

**Enable Dial In:** For a switched line, the line will be enabled for dial-in capability.

## Local LUs

To activate a local LU, select the local LU and choose Activate from the Server menu. To activate all of the local LUs on the list, click the title of the list (Local LUs) and choose Activate from the Server menu.

## Remote LUs

Remote LUs cannot be activated. The remote LUs defined by the configuration file are displayed in the Remote LUs list in the server window.

## Modes

Activating a mode in the Administration program means that session limits are initialized to their defined maximums. To initialize session limits of a mode, select an uninitialized (deactivated) mode and choose Activate from the Server menu.

If the SNASVCMG mode is defined for a remote LU, the remote LU supports parallel sessions. You must activate the SNASVCMG mode first before any user modes. If the remote node is going to initialize session limits on any user mode that supports parallel sessions you must start a CNOS TP for the server. A mode can be activated when the current number of maximum sessions is zero. Mode activity may be monitored by updating and displaying the mode window and noting the number of active sessions, first speakers, and bidders. Once activated, the current number of maximum sessions, first speaker sessions, and bidder sessions are updated. When session limits are initialized, the node tries to bring up the defined number of prebound first speaker sessions.

## Sessions

To activate a session for a mode, select an activated (initialized) mode and choose Activate from the Server menu. For each session activated, the active number of sessions and active first speaker sessions are updated in the mode window. When a session is activated by the remote node, the active number of sessions and active bidder sessions are updated in the mode window only when you choose the Update command from the Server menu.

---

## Deactivating network components and sessions

Activated components and sessions are deactivated from the server window. Components and sessions are generally deactivated in the opposite order that they were activated.



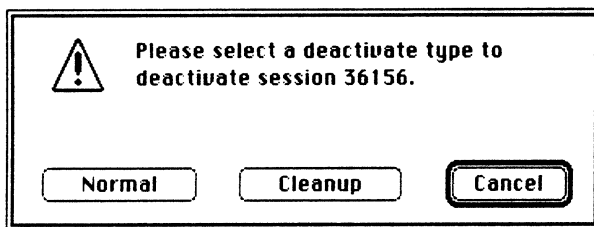
## Order of deactivation

In general, the order of component and session deactivation should be as follows:

1. Sessions
2. User modes
3. SNASVCMG modes
4. Local LUs
5. Stations (if multipoint or switched)
6. Lines.

## Sessions

To deactivate a session, select the session and choose Deactivate from the server menu. To deactivate all of the sessions on the list, click the title of the list (Sessions) and choose deactivate from the Server menu. The dialog box in Figure 12-20 appears to prompt you for the deactivate type.



**Figure 12-20**  
Deactivating a session

**Normal:** The session is to be deactivated only after any current conversation is deallocated.

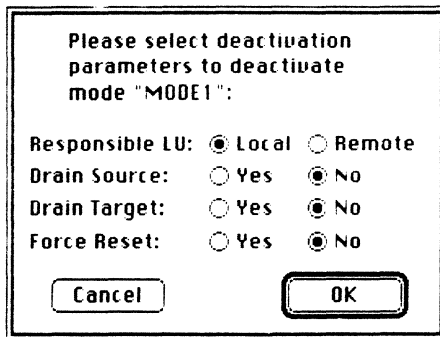
**Cleanup:** The session is to be deactivated immediately, regardless of whether or not a conversation is using it.

For each session deactivated, the active number of sessions is updated in the mode display window. Depending on the polarity of the session, either the active number of first speaker sessions or of bidder sessions may be updated as well. When a session is deactivated, it no longer exists, and therefore disappears from the list of sessions in the server window.

## Modes

Deactivate user modes before deactivating the SNASVCMG mode. A mode can be deactivated when the current number of maximum sessions (as displayed in the mode window) is greater than zero. Deactivating a mode means that its session limits are reset to zero. To deactivate a mode, select an activated (initialized) mode and choose Deactivate from the Server menu. To deactivate all of the modes on the list, click the title of the list (Sessions) and choose Deactivate from the Server menu. The dialog box in Figure 12-21 appears to prompt you for mode deactivation parameters. These are the settings to reset sessions.





**Figure 12-21**  
Deactivating a mode

**Responsible LU:** This setting specifies the LU responsible for deactivating any active sessions. Either the local or remote LU may be chosen.

**Drain Source:** This setting specifies whether or not the source LU can drain its allocation requests. For parallel-session connections, the target LU cannot negotiate this setting. Yes specifies that the source LU can drain its allocation requests. The source LU continues to allocate conversations to the sessions until no requests are awaiting allocation, at which time its draining is ended. No specifies that the source LU cannot drain its allocation requests. All requests currently awaiting allocation, or subsequently issued, at the source LU are rejected.

**Drain Target:** This setting specifies whether or not the target LU can drain its allocation requests. Yes specifies that the target LU can drain its allocation requests. The target LU continues to allocate conversations to the sessions until no requests are awaiting allocation, at which time its draining is ended. All allocation requests issued at the target LU after draining is ended are rejected. No specifies that the target LU cannot drain its allocation requests. All requests currently awaiting allocation, or subsequently issued, at the target LU are rejected.

**Force Reset:** This setting specifies whether or not the source LU is to force the resetting of its session limits when certain error conditions occur that prevent successful exchange of the CNOS request and reply. Yes specifies that the session limit is to be reset upon either successful or unsuccessful completion of the exchange of the CNOS request and reply, except for certain error conditions. No specifies that the session limit is to be reset only upon successful completion of the exchange CNOS request and reply.

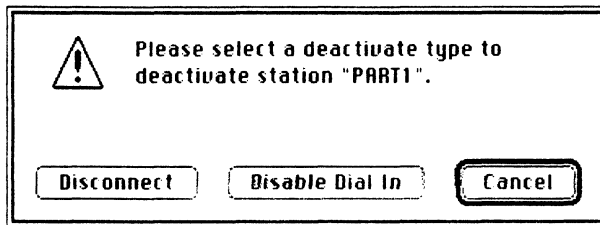
### Local LUs

To deactivate a local LU, select the local LU and choose Deactivate from the Server menu. To deactivate all of the local LUs on the list, click the title of the list (Local LUs) and choose Deactivate from the Server menu.



## Stations

You need to deactivate the station only for a switched or multipoint line. To deactivate a station, select the station and choose Deactivate from the Server menu. To deactivate all of the stations on the list, click the title of the list (Stations) and choose Deactivate from the Server menu. The dialog box in Figure 12-22 appears to prompt you for a deactivate type.



**Figure 12-22**  
Deactivating a station

**Disconnect:** The connection to the station is broken immediately.

**Disable Dial In:** The current connection is maintained. Any future dial-in attempt is not allowed.

## Lines

Deactivating a line terminates the connection. To deactivate a line, select the line and choose Deactivate from the Server menu. To deactivate all of the lines, select the title of the list (Lines) and choose Deactivate from the Server menu.

---

---

## Logging

MacAPPC provides a logging facility to record all internal server messages through a specified MacAPPC server.

Use the Active command from the Log menu to enable the logging functions of the MacAPPC server according to the settings specified with the Settings command of the Log menu. The Show Log command may be used to toggle the display of the logging window on and off. The Clear Log command is used to erase previous MacAPPC server log data. A server must be started for the Active and Settings commands to be active. The log messages are stored in a text file named "Admin Log." The Administration program looks for this file in the System folder and in the folder containing the Administration program. If the Administration program cannot find "Admin Log," the program creates the file.

---

### Log settings options

The logging facility permits filter criteria to be used on log data that is collected and displayed. Criteria include the class, type, and severity of the messages.



To specify the criteria, choose the Settings command from the Log menu. The dialog box in Figure 12-23 appears.

Class: ☒ Node-operator ☒ Logging  
☐ Development ☐ Trace

Type: ☒ Informational ☒ Notification  
☒ Error ☐ Diagnostic

Severity: 90 80 70 60 50 40 30 20 10  
Low Detail Level High

Check for messages every  seconds.

Cancel OK

**Figure 12-23**  
Log settings selection

### Class

**Node-operator:** Node-operator messages are logged.

**Logging:** Logging messages are logged.

**Development:** Development messages are logged.

**Trace:** Trace messages are logged.

### Type

**Informational:** Informational messages are logged.

**Notification:** Notification messages are logged.

**Error:** Error messages are logged.

**Diagnostic:** Diagnostic messages are logged.

### Severity

A severity control is used to select the detail level of logging messages. This control may be adjusted for messages of low to high detail. The severity control is moved similarly to the network display control. To use the severity control, move the cursor to the right end of the shaded area. Drag the cursor to move the control. The control will move to the nearest increment of ten on the scale. The detail level of the logging messages decreases when the control is moved to the left and increases when it is moved to the right.

### Check for messages every \_\_ seconds

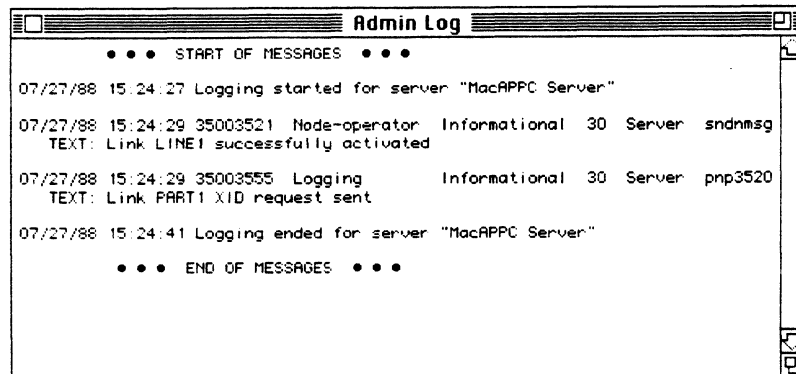
The information in the logging window can be periodically updated by the Administration program. The information update period is specified in seconds. The default setting is 10.



---

## Log Window

To display the log window, choose the Show Log command from the Log menu. The window in Figure 12-24 appears. Log data saved in the current log is displayed and subsequent log data is displayed as it is received. Log messages that are not visible can be brought into view using the scroll bar.



**Figure 12-24**  
The log window

To clear the window, choose Clear Log from the Log menu and click OK when the alert box appears.

---

## Stopping a MacAPPC server

You may terminate the currently displayed server at any time by choosing Stop Server from the Server menu. You can only stop a server from the Macintosh II it resides in. The confirmation alert in Figure 12-25 appears.

---

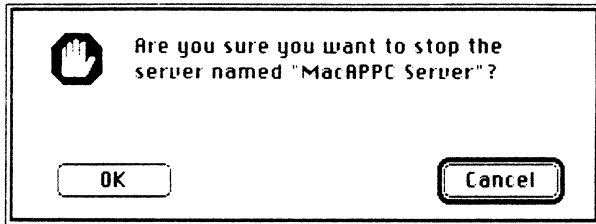
### Important

This command should be used carefully, as it may result in an abrupt termination of the server. Ideally, all sessions and network components should be deactivated first. See the earlier section on deactivating network components and sessions.

---

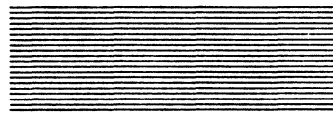


The Administration program stops all CNOS TPs for the server automatically.

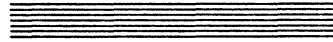


**Figure 12-25**  
Stopping a server





# Appendixes











## **Appendix A**



### **MacAPPC Interface File**

This appendix contains the Pascal interface file for the MacAPPC drivers. For the C and assembly-language interfaces, see APPC.h and APPCEqu.a on the distribution disk.

The information in this file is also available elsewhere in this document. Each routine is documented in the driver chapters (Chapters 4–7). The relevant data structures and data types are found at the end of each driver chapter.



```

{*****}
{ APPC.p -- MacAPPC Interface File }
{ }
{ Pascal Language Interface to the MacAPPC Drivers }
{ Copyright Apple Computer, Inc. 1987,1988 }
{ All rights reserved. }
{*****}

UNIT                                APPC;

INTERFACE

USES

    { $U MemTypes.p }           MemTypes,
    { $U QuickDraw.p }         QuickDraw,
    { $U OSIntf.p }            OSIntf,
    { $U ToolIntf.p }          ToolIntf,
    { $U AppleTalk.p }         AppleTalk;

CONST

{*****}
{ GENERAL CONSTANTS }
{*****}

{ Block sizes }

kAPPCSize      = 2706;    { Maximum APPC Parameter Block size }
kCVSize        = 2706;    { Conversation Parameter Block size }
kCSize         = 192;     { Control Operator Parameter Block size }
kNSize         = 108;     { Node Operator Parameter Block size }
kTPSize        = 1628;    { Transaction Program Parameter Block size }

kTPCBSize      = 3000;    { Transaction Program Control Block size }
kCVCBSize      = 3500;    { Conversation Control Block size }

kLineSize      = 17;      { Maximum Line size }
kSDLCSize      = 17;      { SDLC Line size }

kMsgSize       = 44;      { Message Structure size }
kMsgFieldSize  = 2;       { Message Data Field size }

kMaxPIP        = 256;     { Maximum # of PIPs }
kMaxConvID     = 256;     { Maximum # of ConvIDs }

{ String sizes }

kMaxName       = 8;       { Maximum generic string length }
kMaxTPName     = 64;      { Maximum TP Name length }
kMaxSecName    = 10;      { Maximum Security Field length }
kMaxMapName    = 64;      { Maximum Map Name length }
kMaxPhoneNumber = 20;     { Maximum phone number length }
kMaxLogData    = 200;     { Maximum log data length }
kMaxLUWName    = 17;      { Maximum LUW Name length }
kMaxLUWID      = 6;       { Maximum LUW ID length }
kMaxLUWCorr    = 8;       { Maximum LUW correlator length }
kMaxLUPswd     = 16;      { Maximum LU-LU password length (hex) }
kMaxExchID     = 8;       { Maximum Exchange ID length (hex) }
kMaxCPUID      = 12;      { Maximum CPU ID length (hex) }
kMaxSDLCAddr   = 2;       { Maximum SDLC address length (hex) }

{ Define Operation values }

kIgnoreParam   = (-1);
kFuncNotSupp   = 0;
kFuncSupp      = 1;
kReplaceParam  = 0;
kDeleteParam   = 1;
kAddParam      = 0;
kNextEntry     = 0;

```



```

{*****}
{ APPC CONSTANTS }
{*****}

{ appcOpCode values }

kMCAllocate          = 100;      { Mapped Conversation routines }
kMCConfirm           = 101;
kMCConfirmed         = 102;
kMCDeallocate        = 103;
kMCFlush            = 104;
kMCGetAttributes     = 105;
kMCPostOnReceipt     = 106;
kMCPrepareToReceive  = 107;
kMCReceiveAndWait    = 108;
kMCReceiveImmediate  = 109;
kMCRequestToSend     = 110;
kMCSendData          = 111;
kMCSendError         = 112;
kMCTest              = 113;

kCVBackout           = 114;      { Type Independent routines }
kCVGetType           = 115;
kCVSyncPoint         = 116;
kCVWait              = 117;

kBCAllocate          = 118;      { Basic Conversation routines }
kBCCConfirm          = 119;
kBCCConfirmed        = 120;
kBBCDeallocate       = 121;
kBBCFlush            = 122;
kBBCGetAttributes    = 123;
kBBCPostOnReceipt    = 124;
kBBCPrepareToReceive = 125;
kBBCReceiveAndWait   = 126;
kBBCReceiveImmediate = 127;
kBBCRequestToSend    = 128;
kBBCSendData         = 129;
kBBCSendError        = 130;
kBCTest              = 131;

kCOChangeSessionLimit = 200;      { Control Operator CNOS routines }
kCOInitializeSessionLimit = 202;
kCOProcessSessionLimit = 203;
kCOResetSessionLimit = 204;

kCOActivateSession   = 205;      { Control Operator Session control routines }
}
kCDeactivateSession  = 206;

kCDefineLocalLU       = 207;      { Control Operator LU Definition routines }
kCDefineRemoteLU      = 208;
kCDefineMode          = 209;
kCDefineTP            = 210;
kCDisplayLocalLU      = 211;
kCDisplayRemoteLU     = 212;
kCDisplayMode         = 213;
kCDisplaySession      = 214;
kCDisplayTP           = 215;
kCDelete              = 216;

kNOActivateLine       = 300;      { Node Operator Node Control routines }
kNOActivateLU         = 301;
kNOActivateNode       = 302;
kNOActivateStation    = 303;
kNODeactivateLine     = 305;
kNODeactivateLU       = 306;
kNODeactivateNode     = 307;
kNODeactivateStation  = 308;

kNODefineMessageQueue = 304;      { Node Operator Node Message routines }
kNODisplayMessage     = 310;

```



```

kNODisplayMessageQueue      = 309;

kNODEfineNode               = 311;      { Node Operator Node Definition routines }
kNODEfineCP                 = 312;
kNODEfineLine               = 313;
kNODEfineStation            = 314;
kNODisplayNode              = 316;
kNODisplayCP                = 317;
kNODisplayLine              = 318;
kNODisplayStation           = 319;
kNODElete                   = 320;

kTPAttach                   = 400;      { Transaction Program Connection routines }
kTPDetach                   = 401;

kTPAsciiToEbcDic            = 403;      { Transaction Program Utility routines }
kTPEbcDicToAscii            = 404;

{ appcConvState values }

kNullState                  = 0;
kResetState                 = 1;
kSendState                  = 2;
kReceiveState               = 3;
kConfirmState               = 4;
kConfirmSendState           = 5;
kConfirmDeallocState        = 6;
kDeallocState               = 7;
kDeferState                 = 8;      { not supported }
kSyncPtState                = 9;      { not supported }
kBackedOutState             = 10;     { not supported }

{ *****
{ CONVERSATION CONSTANTS
{ ***** }

{ cvWhatRcvd values }

kNullRcvd                   = 0;
kDataRcvd                   = 1;
kDataComplRcvd              = 2;
kDataIncomplRcvd            = 3;
kLLTruncRcvd                = 4;
kSendRcvd                   = 5;
kConfirmRcvd                 = 6;
kConfirmSendRcvd             = 7;
kConfirmDeallocRcvd          = 8;
kDataTruncRcvd               = 9;
kFMHDataComplRcvd           = 10;
kFMHDataIncomplRcvd         = 11;
kFMHDataTruncRcvd           = 12;
kTakeSyncPtRcvd              = 13;     { not supported }
kTakeSyncPtSendRcvd          = 14;     { not supported }
kTakeSyncPtDeallocRcvd       = 15;     { not supported }

{ cvDeallocType values }

kSyncDealloc                = 1;
kFlushDealloc               = 2;
kAbendProgDealloc           = 3;
kAbendSvcDealloc            = 4;
kAbendTimerDealloc          = 5;
kLocalDealloc               = 6;
kConfirmDealloc              = 7;
kAbendDealloc               = 8;

{ cvPrepToRcvType values }

kFlushRcv                   = 0;
kConfirmRcv                  = 1;
kSyncLevelRcv               = 2;

```



```

{ cvLockType values }

kShortLock          = 0;
kLongLock           = 1;

{ cvFillType values }

kBufferFill         = 0;
kLLFill             = 1;

{ cvSyncType values }

kNoSync             = 0;
kConfirmSync        = 1;
kSyncPtSync         = 2;          { not supported }

{ cvReturnCtl values }

kWhenAllocReturn    = 0;
kDelayAllocReturn   = 1;
kImmedAllocReturn    = 2;

{ cvSecType values }

kNoSec              = 0;
kSameSec            = 1;
kProgSec            = 2;

{ cvConvType values }

kBasicConv          = 0;
kMappedConv         = 1;

{ cvErrorType values }

kSvcError           = 0;
kProgError          = 1;
kAllocError         = 2;          { reserved }

{ cvTestType values }

kPostTest           = 0;
kReqToSendTest      = 1;

{*****}
{ MAPPING FUNCTION CONSTANTS }
{*****}

{ mcpbMapCmd values }

kSendMapping        = 0;
kRcvMapping         = 1;

{ mcpbResult values }

mcNoErr             = 0;
mcErr               = 1;
mcMapNameErr        = 2;
mcDupMapNameErr     = 3;

{ mcpbRcvMode values }

kTruncMode          = 0;
kIncomplMode        = 1;

```



```

{*****}
{ CONTROL OPERATOR CONSTANTS }
{*****}

{ coRespType values }

kSrcResp          = 0;
kTgtResp          = 1;

{ coDeactType values }

kNormalDeact      = 0;
kCleanupDeact     = 1;

{ coSyncType values }

kConfirmModeSync  = 1;
kSyncPtModeSync   = 2;      { not supported }

kNoTPSync         = 1;
kConfirmTPSync    = 2;
kSyncPtTPSync     = 4;      { not supported }

{ coConvType values }

kMappedTPConv     = 0;
kBasicTPConv      = 1;
kEitherTPConv     = 2;

{ coEnableType values }

kEnableTP         = 0;
kDisableTempTP    = 1;
kDisablePermTP    = 2;

{ coPolarType values }

kBidderSess       = 0;
kFirstSpkrSess    = 1;

{ coPrivType values }

kNoPriv           = 0;
kCNOSPriv         = 1;
kSessCtlPriv      = 2;
kDefinePriv       = 4;
kDisplayPriv      = 8;
kSvcTPPriv        = 16;

{ coLclSecAccType and coRmtSecAccType values }

kNoSecAcc         = 0;
kConvSecAcc       = 1;
kVerifSecAcc      = 2;

{ coSecReq values }

kNoSecReq         = 0;
kConvSecReq       = 1;
kProfSecReq       = 2;
kUserSecReq       = 3;
kBothSecReq       = 4;

{ coReinitType values }

kOperInit         = 0;
kPriLUInit        = 1;
kSecLUInit        = 2;
kEitherLUInit     = 3;

```



```

{*****}
{  NODE OPERATOR CONSTANTS                                     }
{*****}

{ noDialType values }

kConnectDial          = 0;
kDialInOnDial         = 1;

kDisconnectDial       = 0;
kDialInOffDial        = 1;

{ noQueueClass values }

kNoChangeQClass       = 0;
kNodeOperMsgsQClass   = 1;
kLogMsgsQClass        = 2;
kDevelMsgsQClass      = 4;
kTraceMsgsQClass      = 8;

{ noQueueType }

kNoChangeQType        = 0;
kInfoMsgsQType        = 1;
kNotifMsgsQType       = 2;
kErrorMsgsQType       = 4;
kDiagMsgsQType        = 8;

{ noQueueSev values }

kNoChangeQSev         = 0;
kReservedQSev         = 1;           { reserved }
kDevelMsgsQSev        = 10;
kLowLevelInfoMsgsQSev = 20;
kNormalInfoMsgsQSev   = 30;
kErrorMsgsQSev        = 40;
kProgErrorsQSev       = 90;

{ noAccessType values }

kSDLCAccess           = 0;

{ noLineStatus values }

kLineReset            = 1;
kLinePendActive       = 2;
kLineActive           = 3;
kLinePendReset        = 4;

{ noLineType values }

kSDLCLine             = 0;

{ noALSStatus values }

kStationReset         = 1;
kStationPendResp      = 2;
kStationPendCont      = 3;
kStationActive        = 4;
kStationPendReset     = 5;
kStationResetPendResp = 6;

```



```

{*****}
{ SDLC LINE CONSTANTS }
{*****}

{ sdlcLineNum values }

kSDLCLine1      = 1;
kSDLCLine2      = 2;
kSDLCLine3      = 3;
kSDLCLine4      = 4;

{ sdlcRoleType values }

kSDLCSecondary  = 0;
kSDLCPrimary    = 1;
kSDLCNegotiable = 2;

{ sdlcConnType values }

kSDLCLeased     = 0;
kSDLCMultiPoint = 1;
kSDLCSwitched   = 2;

{ sdlcDuplexType values }

kSDLCFullDuplex = 0;
kSDLCHalfDuplex = 1;

{ sdlcLineSpeed values }

kSDLC300        = 300;
kSDLC1200       = 1200;
kSDLC2400       = 2400;
kSDLC4800       = 4800;
kSDLC9600       = 9600;
kSDLC19200      = 19200;

{ sdlcNRZIType values }

kSDLCNRZ        = 0;
kSDLCNRZI       = 1;

{*****}
{ TRANSACTION PROGRAM CONSTANTS }
{*****}

{ tpAttachType values }

kLUAttach       = 0;
kSrvrAttach     = 1;
kWaitAttach     = 2;

{ tpDetachType values }

kNormalDetach   = 0;
kAbortDetach    = 1;

{ tpWaitTime values }

kMaxWait        = (-1);    { wait time = forever }
kConfigWait     = 0;       { wait time = local LU wait time }

TYPE

```



```

{*****}
{  MAPPED CONVERSATION PARAMETER BLOCK                                }
{*****}

APPCMCPB =
    mcpbMapCmd      : SignedByte; { MC request }
    mcpbResult      : INTEGER;    { mapper return code }
    mcpbMapName     : StringPtr;  { map name pointer }
    mcpbDataPtr     : Ptr;        { data pointer }
    mcpbDataSize    : INTEGER;    { data length }
    mcpbBuffPtr     : Ptr;        { buffer pointer }
    mcpbBuffSize    : INTEGER;    { buffer length }
    mcpbTransMapName : Boolean;    { map name translation required }
    mcpbFMHdrs      : Boolean;    { FMH data contains FM headers }
    mcpbRcvMode     : SignedByte; { receive mode }
    mcpbAPPCBPBPtr  : Ptr;        { APPC parameter block pointer }
END;

APPCMCPBPtr = ^APPCMCPB;

{*****}
{  LINE STRUCTURES                                                    }
{*****}

APPCLineType = (sdlcLine);

APPCLineRec = RECORD

    CASE APPCLineType OF

        sdlcLine:
        (
            sdlcMaxBTU      : INTEGER; { maximum BTU length }
            sdlcMaxRetry    : INTEGER; { maximum retries }
            sdlcIdleTime    : INTEGER; { idle time before recovery }
            sdlcNPRcvTime   : INTEGER; { non-productive receive time }
            sdlcMaxIFrame   : INTEGER; { maximum I-frames before polling }
            sdlcLineSpeed   : INTEGER; { line speed }
            sdlcLineNum     : SignedByte; { Line Number }
            sdlcRoleType    : SignedByte; { SDLC role }
            sdlcConnType    : SignedByte; { connection type }
            sdlcDuplexType  : SignedByte; { duplex type }
            sdlcNRZIType    : SignedByte; { NRZI support }
        );
    END;

APPCLineRecPtr = ^APPCLineRec;

{*****}
{  NODE MESSAGES STRUCTURES                                          }
{*****}

APPCNOMsg = RECORD
    msgDataSize      : INTEGER; { Length of message unit }
    msgMonth          : SignedByte; { Month }
    msgDay            : SignedByte; { Day }
    msgYear           : SignedByte; { Year }
    msgHour           : SignedByte; { Hour }
    msgMinute         : SignedByte; { Minute }
    msgSecond         : SignedByte; { Second }
    msgID             : LONGINT; { Message ID }
    msgConst          : SignedByte; { Structure constant }
    msgClass          : SignedByte; { Class of message }
    msgType           : SignedByte; { Type of message }
    msgSevType        : SignedByte; { Severity of message }
    msgMUType         : SignedByte; { MU type (kLogMU,kDumpMU) }
    msgCorrID         : INTEGER; { Correlation number }
    msgSenseCode      : INTEGER; { Send Check Sense Code }
    msgSenseExt       : INTEGER; { Check Sense Extension }

```



```

    msgProcID      :    INTEGER;      { Procedure ID }
    msgProcName    :    PACKED ARRAY[1..16] OF CHAR; { Name of source module }
    msgFieldCount  :    INTEGER;      { Length of data fields }
END;

APPCNOMsgField =
    msgFieldType   :    SignedByte;   { Type of data }
    msgFieldDataSize :    SignedByte;   { Length of field that follows }
END;

APPCNOMsgPtr =
APPCNOMsgFieldPtr =
    ^APPCNOMsg;
    ^APPCNOMsgField;

{ ***** }
{  PROTOCOL STRUCTURES  }
{ ***** }

{ APPC Parameter Block }

APPCParamType      =      (cvParam,coParam,noParam,tpParam);

APPCParamBlock =
    RECORD
        qLink      :    QElemPtr;      { DRVR QElem pointer }
        qType      :    INTEGER;        { DRVR queue type }
        ioTrap     :    INTEGER;        { DRVR IO trap }
        ioCmdAddr  :    Ptr;            { DRVR IO command pointer }
        ioCompletion :    ProcPtr;      { DRVR IO completion routine pointer }
        ioResult   :    OSerr;          { DRVR IO result }
        ioNamePtr  :    StringPtr;      { DRVR IO name pointer }
        ioVRefNum  :    INTEGER;        { DRVR IO volume refNum }
        appcRefNum :    INTEGER;        { APPC driver refNum }
        appcOpCode :    INTEGER;        { APPC type of call }
        appcHiResult :    INTEGER;      { APPC major result code }
        appcLoResult :    INTEGER;      { APPC minor result code }
        appcConvState :    SignedByte;  { APPC conversation state }
        appcUserRef :    LONGINT;       { for your use }
    END;

CASE
    APPCParamType OF

        cvParam:
        (
            cvTPCBPtr      :    Ptr;      { TPCB pointer }
            cvCVCBPtr      :    Ptr;      { CVCB pointer }
            cvPIPBufPtr    :    Ptr;      { PIP buffer pointer }
            cvPIPBufSize   :    INTEGER;   { PIP buffer size }
            cvMapBufPtr    :    Ptr;      { mapped conversation buffer pointer }
            cvMapBufSize   :    INTEGER;   { mapped conversation buffer size }
            cvConvID       :    LONGINT;   { conversation ID }
            cvProgID       :    LONGINT;   { transaction program ID }
            cvRmtLUName    :    StringPtr; { remote LU name pointer }
            cvFullRmtLUName :    StringPtr; { fully qualified RLU name pointer }
            cvFullLclLUName :    StringPtr; { fully qualified LLU name pointer }
            cvModeName     :    StringPtr; { mode name pointer }
            cvRmtProgName  :    StringPtr; { remote program name pointer }
            cvUserName     :    StringPtr; { user name pointer }
            cvUserPswd     :    StringPtr; { user password pointer }
            cvUserProf     :    StringPtr; { user profile pointer }
            cvLUWName      :    StringPtr; { Logical Unit of Work LU name pointer }
            cvLUWID        :    StringPtr; { LUW identifier pointer }
            cvLUWCorr      :    StringPtr; { LUW conversation correlator pointer }
            cvLUWSeq       :    INTEGER;   { LUW sequence number }
            cvDataPtr      :    Ptr;      { data buffer pointer }
            cvDataSize     :    INTEGER;   { data buffer size }
            cvMapName      :    StringPtr; { map name pointer }
            cvMapProc      :    ProcPtr;   { mapping procedure pointer }
            cvSenseData    :    LONGINT;   { reserved }
            cvReqToSendRcvd :    Boolean;   { request to send received }
            cvFMHdrrs      :    Boolean;   { FM headers in data record }
            cvWhatRcvd     :    SignedByte; { what was received }
            cvDeallocType  :    SignedByte; { deallocation type }
            cvPrepToRcvType :    SignedByte; { prepare to receive type }
            cvLockType     :    SignedByte; { prepare to receive lock }
        )
    END;

```



```

cvFillType      : SignedByte; { logical record receive }
cvSyncType      : SignedByte; { synchronization level }
cvReturnCtl     : SignedByte; { allocate return control }
cvSecType       : SignedByte; { security type }
cvConvType      : SignedByte; { conversation type }
cvErrorType     : SignedByte; { send error type }
cvTestType      : SignedByte; { test type }
cvPIPUsed       : Boolean;     { program parameters used }
cvCVCBIndex     : INTEGER;     { CVCB pointer list index }
cvCVCBList      : ARRAY[1..kMaxCVCB] OF Ptr; { list of CVCB ptrs }
cvPIPPtr        : ARRAY[1..kMaxPIP] OF Ptr; { array of PIP ptrs }
cvPIPSize       : ARRAY[1..kMaxPIP] OF INTEGER; { array of PIP sizes }
);

coParam:
(
coTPCBPtr       : Ptr;         { TPCB pointer }
coCVCBPtr       : Ptr;         { CVCB pointer }
coLclLUName     : StringPtr;   { local LU name pointer }
coRmtLUName     : StringPtr;   { remote LU name pointer }
coModeName      : StringPtr;   { mode name pointer }
coLclProgName   : StringPtr;   { local TP name pointer }
coNetName       : StringPtr;   { network name pointer }
coNetQual       : StringPtr;   { network qualifier pointer }
coCPName        : StringPtr;   { control point name pointer }
coALSName       : StringPtr;   { adjacent link station name pointer }
coCNOSALSName   : StringPtr;   { CNOS station name pointer }
coUserName      : StringPtr;   { user name pointer }
coUserPswd      : StringPtr;   { user password pointer }
coUserProf      : StringPtr;   { user profile pointer }
coLUPswd        : StringPtr;   { LU-LU password pointer }
coConvID        : LONGINT;     { conversation ID }
coProgID        : LONGINT;     { transaction program ID }
coSessID        : LONGINT;     { session ID }
coLclLUID       : SignedByte;  { local LU ID }
coWaitTime      : INTEGER;     { wait time in secs }
coDrainSrc      : Boolean;     { reset drain source }
coDrainTgt      : Boolean;     { reset drain target }
coForceRst      : Boolean;     { force reset }
coDrainLclLU    : Boolean;     { drain local LU }
coDrainRmtLU    : Boolean;     { drain remote LU }
coLUActive      : Boolean;     { LU activation status }
coNextLclLUName : SignedByte;  { display next local LU }
coNextRmtLUName : SignedByte;  { display next remote LU }
coNextModeName  : SignedByte;  { display next mode }
coNextLclProgName : SignedByte; { display next local TP }
coNextUserName  : SignedByte;  { display next user }
coNextSessID    : SignedByte;  { display next session ID }
coNetNameOp     : SignedByte;  { network name operation }
coNetQualOp     : SignedByte;  { network qualifier operation }
coLUPswdOp      : SignedByte;  { password operation }
coSecOp         : SignedByte;  { security operation }
coLUWSupp       : SignedByte;  { LUW support }
coParSess       : SignedByte;  { parallel session support }
coQueueBINDs    : SignedByte;  { queue BINDs }
coDataMapping    : SignedByte; { data mapping support }
coFMHDataSupp   : SignedByte;  { FMH data support }
coPIPReq        : SignedByte;  { PIP required }
coPIPCheck      : SignedByte;  { check PIP count }
coPIPCount      : INTEGER;     { PIP count }
coSessCrypt     : SignedByte;  { session level cryptography }
coBlankMode     : SignedByte;  { blank mode option }
coQueueINITs    : SignedByte;  { queue session-initiation requests }
coConvSecType   : SignedByte;  { conversation level security }
coSyncType      : SignedByte;  { synchronization level }
coConvType      : SignedByte;  { conversation type }
coDeactType     : SignedByte;  { deactivate type }
coEnableType    : SignedByte;  { enable status type }
coRespType      : SignedByte;  { deactivate responsibility }
coPolarType     : SignedByte;  { session polarity }
coPrivType      : SignedByte;  { privilege }
coLclSecAcc     : SignedByte;  { local LU security acceptance }

```



```

coRmtSecAcc      : SignedByte; { remote LU security acceptance }
coSecReq         : SignedByte; { security required }
coReinitType     : SignedByte; { single session reinitiation }
coTermCount      : INTEGER;    { termination count }
coMaxTP          : INTEGER;    { maximum attached TPs }
coSendPacing     : INTEGER;    { send pacing window }
coRcvPacing      : INTEGER;    { receive pacing window }
coMaxRUHiBound   : INTEGER;    { maximum RU upper bound }
coMaxRULoBound   : INTEGER;    { maximum RU lower bound }
coDefLUMaxSess   : INTEGER;    { defined max LU sessions }
coActLUSess      : INTEGER;    { active LU sessions }
coDefMaxSess     : INTEGER;    { defined max sessions }
coCurMaxSess     : INTEGER;    { current max sessions }
coActSess        : INTEGER;    { active sessions }
coDefMinFirstSpkrs : INTEGER;  { defined min first speakers }
coCurMinFirstSpkrs : INTEGER;  { current min first speakers }
coActFirstSpkrs  : INTEGER;    { active first speakers }
coDefMinBdrs     : INTEGER;    { defined min bidders }
coCurMinBdrs    : INTEGER;    { current min bidders }
coActBdrs        : INTEGER;    { active bidders }
coDefPBFfirstSpkrs : INTEGER;  { prebound first speakers }
);

noParam:
(
noTPCBPtr       : Ptr;         { TPCB pointer }
noLclLUName     : StringPtr;   { local LU name pointer }
noALSName       : StringPtr;   { adjacent link station name pointer }
noLineName      : StringPtr;   { line name pointer }
noCPName        : StringPtr;   { control point name pointer }
noQueueName     : StringPtr;   { queue name pointer }
noPhoneNumber   : StringPtr;   { phone number pointer }
noALSAddr       : StringPtr;   { ALS Address pointer }
noExchID        : StringPtr;   { exchange ID pointer }
noCPUID         : StringPtr;   { CPU ID pointer }
noCorrID        : INTEGER;     { correlation ID }
noDataPtr       : Ptr;         { data buffer pointer }
noDataSize      : INTEGER;     { data buffer length }
noMonTimer      : INTEGER;     { wakeup timer for monitor }
noAccessType    : SignedByte;  { Access Type }
noStopSrvr     : Boolean;      { halt server }
noDialType      : SignedByte;  { dial type }
noQueueEnable   : Boolean;     { queue enabled }
noQueueClass    : SignedByte;  { queue class }
noQueueType     : SignedByte;  { queue type }
noQueueSev      : SignedByte;  { queue severity level }
noWaitForMsg    : Boolean;     { block if no message in queue }
noNextALSName   : SignedByte;  { next adjacent link station name }
noNextLineName  : SignedByte;  { next line name }
noNextCPName    : SignedByte;  { next CP name }
noNodeMsgs     : SignedByte;  { enable node messages }
noLogMsgs       : SignedByte;  { enable logging messages }
noLineStatus    : SignedByte;  { line status }
noLineType     : SignedByte;  { line type }
noALSStatus     : SignedByte;  { station status }
noLinePtr       : Ptr;         { line structure pointer }
);

tpParam:
(
tpTPCBPtr       : Ptr;         { TPCB pointer }
tpCVCBPtr       : Ptr;         { CVCB pointer }
tpPIPBuffPtr    : Ptr;         { PIP buffer pointer }
tpPIPBuffSize   : INTEGER;     { PIP buffer size }
tpMapBuffPtr    : Ptr;         { mapped conversation buffer pointer }
tpMapBuffSize   : INTEGER;     { mapped conversation buffer size }
tpLclLUName     : StringPtr;   { local LU name pointer }
tpLclProgName   : StringPtr;   { local program name pointer }
tpSrvrEntityPtr : EntityPtr;   { server entity pointer }
tpConvID        : LONGINT;     { conversation ID }
tpProgID        : LONGINT;     { transaction program ID }
tpWaitTime      : INTEGER;     { wait time in seconds }

```



```

tpAttachType      : SignedByte; { attach type }
tpDetachType      : SignedByte; { detach type }
tpMapProc         : ProcPtr;     { mapping procedure pointer }
tpDataPtr         : Ptr;         { data buffer pointer }
tpDataSize        : INTEGER;     { data buffer size }
tpPIPPtr          : ARRAY[1..kMaxPIP] OF Ptr; { array of PIP ptrs }
tpPIPSize         : ARRAY[1..kMaxPIP] OF INTEGER; { array of PIP sizes }
);
END;

APPCCParamBlockPtr = ^APPCCParamBlock;
APPCCParamBlockHandle = ^APPCCParamBlockPtr;

END.

```









## **Appendix B**



### **MacAPPC Errors File**

This appendix contains the Pascal interface errors file for the MacAPPC drivers. For the C and assembly-language error interfaces, see APPCErrors.h and APPCErrorsEqu.a on the distribution disk.

The information in this file is also available elsewhere in this document. Appendix C "MacAPPC Result Codes" gives a complete listing of all values for `appcHiResult` and `appcLoResult` and provides a detailed explanation of the meaning of each combination of codes.



```

{*****}
{
  APPCErrors.p -- MacAPPC Error Values
}
{
  Pascal Language Interface to the MacAPPC Drivers
}
{
  Copyright Apple Computer, Inc. 1986-1988
}
{
  All rights reserved.
}
{*****}

UNIT                APPCErrors;

INTERFACE

CONST

{*****}
{
  APPC routine return codes
}
{*****}

appcNoErr           = 0;    { successful completion }
appcExec            = 1;    { asynchronous execution active }
appcFail            = (-5000); { error, see appcResult field }

{*****}
{
  Major result codes := appcHiResult
}
{*****}

{noErr              = 0; } { Completed Normally }
usageErr            = 1;   { Aborted, Usage Error }
badComplErr         = 2;   { Unsuccessful }
stateErr            = 3;   { Aborted, State Error }
allocErr            = 5;   { Allocation Error }
progErr             = 7;   { Program Error }
deallocErr          = 9;   { Deallocated }
ctlOpErr            = 10;  { Control Operator Error }
nodeOpErr           = 11;  { Node Operator Error }

{*****}
{
  Minor result codes := appcLoResult
}
{*****}

{ noErr }

normalCompl         = 0;    { OK as Specified }
negotCompl          = 1;    { OK as Negotiated }
dataAvail           = 3;    { TEST - DATA }
ctlAvail            = 4;    { TEST - NO_DATA }
badErr              = 86;   { Default/undefined }

{ usageErr }

notAttachErr        = 1;    { program not ATTACHed }
dupAttachErr        = 2;    { duplicate ATTACH attempted }
unkConvErr          = 3;    { invalid conversation (resource unknown) }
badAttachErr        = 4;    { routine not valid for ATTACH type }
tpcbErr             = 5;    { TPCB not set }
tpNameErr           = 6;    { TP name invalid }
lookupErr           = 7;    { server lookup failed }
connectErr          = 8;    { server connect failed }
noParamErr          = 9;    { null struc parm passed }
srvrEntityErr       = 10;   { ATTACH: bad server entity }
lclLUNameErr        = 11;   { ATTACH: LU name required }
attachTypeErr       = 12;   { ATTACH: type invalid }
unkLclLUNameErr     = 13;   { ATTACH: LU unknown }
availLUErr          = 14;   { ATTACH: LU not available }

```



lclProgNameErr	= 15;	{ ATTACH: TP not configured }
badConvErr	= 18;	{ ATTACH: invalid conversation }
badWaitTimeErr	= 19;	{ ATTACH: invalid wait time }
procSpaceErr	= 20;	{ ATTACH: not enough space }
procLimitErr	= 21;	{ ATTACH: process limit reached }
cvcbErr	= 25;	{ CVCB not set }
dataErr	= 26;	{ data pointer or size not set }
detachTypeErr	= 27;	{ DETACH: type invalid }
rmtLUNameErr	= 30;	{ ALLOC: RLU required }
modeNameErr	= 31;	{ ALLOC: MODE required }
rmtProgNameErr	= 32;	{ ALLOC: TP required }
unkRmtLUNameErr	= 33;	{ ALLOC: RLU unknown }
unkModeNameErr	= 34;	{ ALLOC: MODE unknown }
convTypeErr	= 35;	{ ALLOC: type invalid }
mapConvErr	= 36;	{ ALLOC: mapped conv n/a }
basicConvErr	= 37;	{ ALLOC: basic conv n/a }
returnCtlErr	= 38;	{ ALLOC: when invalid }
delayAllocErr	= 39;	{ ALLOC: delayed alc n/a }
immedAllocErr	= 40;	{ ALLOC: immediate alc n/a }
syncTypeErr	= 41;	{ ALLOC: sync invalid }
badSyncErr	= 42;	{ ALLOC: sync n/a }
pipErr	= 43;	{ ALLOC: PIP n/a }
badPipErr	= 44;	{ ALLOC: PIP maxsize exceeded }
badSecTypeErr	= 45;	{ ALLOC: sec invalid }
secTypeErr	= 46;	{ ALLOC: requested security n/a }
userNameFmtErr	= 47;	{ ALLOC: bad user format }
userPswdFmtErr	= 48;	{ ALLOC: bad pass format }
pipBuffErr	= 49;	{ ALLOC: bad PIP buffer }
mapBuffErr	= 50;	{ ALLOC: bad mapped conversation buffer }
badSyncTypeErr	= 60;	{ CONFIRM(ED): bad sync-level }
sendReqErr	= 61;	{ write to server failed }
recvRspErr	= 62;	{ read from server failed }
verbReqErr	= 63;	{ internal verb request failed }
deallocTypeErr	= 70;	{ DEALLOC: type invalid }
prepToRcvTypeErr	= 80;	{ PREP_TO_RCV: type invalid }
lockTypeErr	= 81;	{ PREP_TO_RCV: lock invalid }
fillTypeErr	= 90;	{ RCV_WAIT,POST_RCPT: fill invalid }
dataSizeErr	= 91;	{ data size is invalid }
badLLErr	= 100;	{ SEND_DATA: invalid LL }
badPSHdrLLErr	= 101;	{ SEND_DATA: invalid LL - PS hdr }
errorTypeErr	= 110;	{ SEND_ERR: type invalid }
badErrorTypeErr	= 111;	{ SEND_ERR: error type reserved }
logDataErr	= 112;	{ SEND_ERR,DEALLOC: error log n/a }
cvcbIndexErr	= 120;	{ WAIT: count < 0 or > kMaxCVCB invalid }
cvcbListErr	= 121;	{ WAIT: invalid CVCB on list }
badPostConvErr	= 122;	{ WAIT: listed cnv not posting }
postConvErr	= 123;	{ WAIT: no posting conv's }
badWaitConvErr	= 126;	{ WAIT: conv posted not in CVCB list }
testTypeErr	= 130;	{ TEST: invalid type }
postActErr	= 131;	{ TEST: posting not active }
notAuthCOErr	= 300;	{ not authorized for CO routines }
sessLimitErr	= 301;	{ Requested Limits are invalid }
modeSVCMGRErr	= 302;	{ SVCMgr Mode not initialized }
zeroSessLimitErr	= 310;	{ Session limit 0 invalid }
respTypeErr	= 311;	{ Responsible value invalid }
sessSumErr	= 312;	{ MAX less than MINF + MINB }
deactTypeErr	= 313;	{ Illegal immed param on dctses }
drainTgtErr	= 314;	{ Illegal dtrg param on rstsl }
drainSrcErr	= 315;	{ illegal dsrc param on rstsl }
forceRstErr	= 316;	{ Illegal force param on rstsl }
badSessIDErr	= 320;	{ Session ID invalid }
nextSessErr	= 321;	{ no next session found }
nameErr	= 400;	{ NO - name too long }
lineErr	= 401;	{ NO - LINE required }
luErr	= 402;	{ NO - LU required }
puErr	= 403;	{ NO - PU required }
dialTypeErr	= 404;	{ NO - invalid dial }
queueNameErr	= 405;	{ NO - bad queue name }
sevTypeErr	= 411;	{ NO - bad severity }
invLclLUNameErr	= 501;	{ DEF - Invalid Local LU }
invRmtLUNameErr	= 502;	{ DEF - Invalid Remote LU }



```

invModeNameErr      = 503; { DEF - Invalid Mode }
alsNameErr          = 504; { DEF - Invalid ALS }
sendPacingErr       = 505; { DEF - Inv Send Pacing Window }
recvPacingErr       = 506; { DEF - Inv Rec Pacing Window }
maxRUHiBoundErr     = 507; { DEF - Inv Max RU Upper Bound }
maxRULoBoundErr     = 508; { DEF - Inv Max RU Lower Bound }
invSyncTypeErr      = 509; { DEF - Inv Sync Level Option }
reinitTypeErr       = 510; { DEF - Inv Reinit Option }
sessCryptErr        = 511; { DEF - Inv Crypt Option }
defMaxSessErr       = 512; { DEF - Inv Max Number Sess }
defMinFirstSpkrsErr = 513; { DEF - Inv Min Num Fst Spk }
defPBFFirstSpkrsErr = 514; { DEF - Inv Min Num Fst Preb }
queueBINDsErr       = 515; { DEF - Inv Bind Queue Option }
blankModeErr        = 516; { DEF - Inv Blank Mode Option }
netNameOpErr        = 517; { DEF - Inv Network Name Oper }
netQualOpErr        = 518; { DEF - Inv Network Qual Oper }
netNameErr          = 519; { DEF - Inv Network Name }
netQualErr          = 520; { DEF - Inv Network Qualifier }
queueINITsErr       = 521; { DEF - Inv Queue Init Option }
parSessErr          = 522; { DEF - Inv Parallel Ses Opt }
cnosALSNameErr      = 523; { DEF - Inv CNOS ALS }
luPswdErr           = 524; { DEF - Inv LU Password }
secAccErr           = 525; { DEF - Inv Security Acpt Opt }
luPswdOpErr         = 526; { DEF - Inv Password Operation }
defLUMaxSessErr     = 527; { DEF - Inv LU Session Limits }
convSecErr          = 528; { DEF - Inv Conversation Sec }
secOpErr            = 529; { DEF - Inv Security Operation }
userNameErr         = 530; { DEF - Inv User ID }
userPswdErr         = 531; { DEF - Inv Password }
userProfErr         = 532; { DEF - Inv Profile }
waitTimeErr         = 533; { DEF - Inv Wait }
maxTPErr            = 534; { DEF - Inv Max Number of TPs }
luIDErr             = 535; { DEF - Inv LU ID Number }
tpNameDefErr        = 536; { DEF - Inv Trans Prog Name }
luActiveErr         = 537; { DEF - Inv Status }
invConvTypeErr      = 538; { DEF - Inv Conversation Type }
secReqErr           = 539; { DEF - Inv Security Required }
pipReqErr           = 540; { DEF - Inv PIP Option }
pipCountErr         = 541; { DEF - Inv PIP Number }
pipCheckErr         = 542; { DEF - Inv PIP Check }
dataMapErr          = 543; { DEF - Inv Data Mapping }
fmhDataSuppErr      = 544; { DEF - Inv FMH }
privTypeErr         = 545; { DEF - Inv Privilege }
luwSuppErr          = 546; { DEF - Inv LUW Indicator }

unkLUErr            = 550; { DEF - LU Unknown }
unkRmtErr           = 551; { DEF - Partner Unknown }
unkALSErr           = 552; { DEF - ALS Unknown }
defSpaceErr         = 553; { DEF - Not enough space }
unkRmtPUErr         = 554; { DEF - Remote PU Unknown }
unkLineErr          = 555; { DEF - Line Unknown }
unkLSCBErr          = 556; { DEF - LSCB not found }
lineActiveErr       = 557; { DEF Sta - Associated Line not inactive }
unkALCBErr          = 558; { DEF - ALCB not found }

modeParamErr        = 560; { DEF Mode - Invalid Parameter }
maxRUBoundErr       = 561; { DEF Mode - Lower Bound > Upper Bound }
reinitErr           = 562; { DEF Mode - Single Session Reinit error }
lclStationErr       = 563; { DEF Mode - lcl station & ptnr != lcl }
minFirstSpkrsErr    = 564; { DEF Mode - minf > maxs }
pbFirstSpkrsErr     = 565; { DEF Mode - mipf > minf }
badMaxSessErr       = 566; { DEF Mode - maxsess not error }
badBlankModeErr     = 567; { DEF Mode - blank mode already exists }

cnosALSErr          = 570; { DEF RLU - CNOS ALS and parsess support error }
}
parSessMaxSessErr   = 571; { DEF RLU - parsess and maxsess error }
lclALSBadNameErr    = 572; { DEF RLU - ALS is lcl and rnam != lnam }
rmtALSBadNameErr    = 573; { DEF RLU - ALS not local & rnam == lnam }
parSessReinitErr    = 574; { DEF RLU - bad reinit option for parsess }
unkCNOSALSErr       = 575; { DEF RLU - CNOS als unknown }
badNetNameErr       = 576; { DEF RLU - null net name already exists }

```



```

parSessPUTypeErr      = 577; { DEF RLU - parsess and PU type error }

lclPUErr              = 580; { DEF LLU - drcb for local PU !exist }
maxSessLimitErr       = 581; { DEF LLU - sess limits > def LU sess limits
}
initNetNameErr        = 582; { DEF LLU - net name spec after inits1 }
unkLUSecErr           = 583; { DEF LLU - sec parms to be del !found }
badLUIDErr            = 584; { DEF LLU - LU id already specified }
initLUIDErr           = 585; { DEF LLU - LU id must be specified }
luIDUpdateErr         = 586; { DEF LLU - LU id can't be updated }

badSecReqErr          = 590; { DEF TP - sec access parms != sec req }
unkTPSecErr           = 591; { DEF TP - sec parms to be del !found }

accessTypeErr         = 595; { DEF - Inv Access Type }
lineNumErr            = 596; { DEF - Inv Line Number }
eaErr                 = 600; { DEF - no ea available }
exchIDErr             = 601; { DEF - Inv Exchange ID }
invDevNameErr         = 602; { DEF - Inv Master Device }
monTimerErr           = 603; { DEF - Inv Monitor Timer }
nodeMsgsErr           = 604; { DEF - Inv NOOP Messages }
logMsgsErr            = 605; { DEF - Inv LOG Messages }
debugMsgsErr          = 606; { DEF - Inv Debug Messages }
cpNameErr             = 607; { DEF - Inv PU Name }
cpuIDErr              = 608; { DEF - Inv CPU ID }
lineNameErr           = 609; { DEF - Inv LINE Name }
lineTypeErr           = 610; { DEF - Inv LINE Type }
sdlcRoleErr           = 612; { DEF - Inv SDLC Role }
connTypeErr           = 613; { DEF - Inv Connection Type }
nrziTypeErr           = 614; { DEF - Inv NRZI }
duplexTypeErr         = 615; { DEF - Inv Half Duplex }
maxBTUErr             = 616; { DEF - Inv Max BTU }
maxRetrysErr          = 617; { DEF - Inv Max Retries }
idleTimeErr           = 618; { DEF - Inv Idle }
npRcvTimeErr          = 619; { DEF - Inv Nonprod Rcv Time }
maxIFramesErr         = 620; { DEF - Inv Max I-Frames }
lineSpeedErr          = 621; { DEF - Inv Rate }
alsAddrErr            = 622; { DEF - Inv SDLC Address }
phoneNumberErr        = 623; { DEF - Inv Phone Name }
maxSOTErr             = 624; { DEF - Exceeded maxsot }
badSDLCRoleErr        = 626; { DEF Line - multidrop and role
!primary }
badLineTypeErr        = 627; { DEF Sta - 1 sta def for leased line }
badLineErr            = 628; { DEF Line - LINE already exists }

unkLineNameErr        = 630; { DEF Sta - LINE Not Specified }
unkPUErr              = 631; { DEF Sta - PU Not Specified }
badStationErr         = 632; { DEF Sta - Station already exists }
badLineNameErr        = 634; { DSP Line - bad LINE name }
endLineListErr        = 635; { DSP Line - end of LINE list }
noLineListErr         = 636; { DSP Line - no LINES defined }
badPUNameErr          = 637; { DSP RPU - bad PU name }
endPUListErr          = 638; { DSP RPU - end of PU list }
noPUListErr           = 639; { DSP RPU - no PUs defined }

unkModeNameDelErr     = 640; { DEL - MODE unknown }
unkProgNameDelErr     = 641; { DEL - TP unknown }
objInUseErr           = 642; { DEL - object is in use }
badLclLUNameDelErr    = 646; { DEL - local LU name not specified }
badRmtLUNameDelErr    = 647; { DEL - remote LU name not specified }

delNameErr            = 648; { DEL - no parms specified }
xidOrCPIDErr          = 650; { DEF RPU - neither xid | cpid specified }
nodeIDErr             = 651; { DEF RPU - both xid and cpid specified }

badUserNameErr        = 660; { DSP LLU - user ID invalid }
endSecListErr         = 661; { DSP LLU - end of security list }
unkUserNameErr        = 662; { No user ID found }
badUserProfErr        = 663; { DSP LLU - profile invalid }
badLUNameErr          = 664; { DSP LLU - bad LU name }
endLUListErr          = 665; { DSP LLU - end of LU list }
luNameErr             = 666; { DSP LLU - no LUs defined }

```



```

badRmtLUNameErr      = 667; { DSP RLU - bad RLU name }
endRmtLUListErr      = 668; { DSP RLU - end of RLU list }
noRmtLUListErr       = 669; { DSP RLU - no RLUs defined }
badModeNameErr       = 670; { DSP Mode - bad MODE name }
endModeListErr       = 671; { DSP Mode - end of MODE list }
noModeListErr        = 672; { DSP Mode - no MODEs defined }
badTPNameErr         = 673; { DSP TP - bad TP name }
endTPListErr         = 674; { DSP TP - end of TP list }
noTPListErr          = 675; { DSP TP - no TPs defined }
noNetNameListErr     = 676; { DSP TP - no network name }

srvrSpaceErr         = 840; { server out of request space }
srvrNotActiveErr     = 850; { server is not active }
srvrCrashedErr       = 860; { server has crashed }
timeOutNoRecErr      = 870; { Time out - Request not recovered }
timeOutRecErr        = 880; { Time out - Request recovered }
timeOutQueFullErr    = 890; { Time out - Msgq full }
logicErr             = 900; { logic error }

{ badComplErr }

rcvImmErr            = 1;  { receive immediate unsuccessful }
noDataErr            = 2;  { Not posted. No data or info }
noReqToSendErr       = 3;  { No request to send }

{ stateErr }

convStateErr         = 1;  { Conversation state error }
logicalRecErr        = 2;  { Logical Record stateerror }
waitConvStateErr     = 3;  { conv for WAIT not RECV state }

{ allocErr }

tpNoRetryErr         = 1;  { TP not available - no retry }
tpRetryErr           = 2;  { TP not available - retry }
convTypeMatchErr     = 3;  { Conversation type mismatch }
pipSuppErr           = 4;  { PIP data not supported }
pipSpecErr           = 5;  { Error in PIP specification }
secInfoErr           = 6;  { Error in Security Information }
syncTypeSuppErr      = 7;  { Program doesn't support sync }
unkProgNameErr       = 8;  { TP not recognized }
allocNoRetryErr      = 9;  { Allocation Failure - no retry }
allocRetryErr        = 10; { Allocation Failure - retry }
immSessErr           = 11; { immediate session not available }
rsrcFailErr          = 12; { local resource failure }
badAllocErr          = 13; { ALLOCATION_ERROR }
allocSubErr          = 16; { ALLOCATION_ERROR (subcoded) }
allocGenErr          = 22; { Generic allocation failure }
unkAllocErr          = 99; { unknown allocation error }

{ progErr }

progNoTruncErr       = 1;  { Program Error - No Truncation }
progLLTruncErr       = 2;  { Program Error - LL Truncated }
progPurgingErr       = 3;  { Program Error - Purging }
svcNoTruncErr        = 11; { Service Error - No truncation }
svcLLTruncErr        = 12; { Service Error - LL Truncated }
svcPurgingErr        = 13; { Service Error - Purging }
fmhSuppErr           = 20; { FMH_DATA_NOT_SUPPORTED }
mapSuppErr           = 21; { MAPPING_NOT_SUPPORTED }
mapNameErr           = 22; { MAP_NOT_FOUND }
mapProcErr           = 23; { MAP_EXECUTION_FAILURE }
dupMapNameErr        = 24; { DUPLICATE_MAP_NAME }

{ deallocErr }

normDeallocErr       = 0;  { Deallocate - Normal }
abendProgErr         = 1;  { Deallocate - Abend Program }
abendSvcErr          = 2;  { Deallocate - Abend Service }
abendTimerErr        = 3;  { Deallocate - Abend Timer }
rsrcErr              = 4;  { Resource Failure }
abendErr             = 5;  { Deallocate - Abend }

```



```

{ ctloPerr }

limitsNotZeroErr      = 63; { Limits are not zero }
limitsTooBigErr       = 64; { Requested limits exceed configuration }
limitsSumTooBigErr    = 65; { Minimums exceed max.session }
badSNASVCMGLimitsErr = 66; { Invalid SNAVCMG limits }
modeSNASVCMGinitErr  = 67; { SNASVCMG MODE not initialized }
limitsClosedErr       = 68; { Mode limits are closed }
noCNOSerr             = 69; { CHGSL not valid }
userModesErr          = 70; { SNASVCMG MODE can't be reset }
raceCNOSerr           = 71; { CNOS race at remote - they won }
unkModeErr            = 72; { Partner doesn't recognize MODE }
localCNOSerr          = 73; { CNOS is in process locally }
allocCNOSerr          = 74; { CNOS allocation error }
rsrccNOSerr           = 75; { CNOS resource failure }
noSessSpaceErr        = 76; { Not enough space for session }
rmtLUActErr           = 77; { Partner LU not active }
modeParmErr           = 78; { Mode configurations don't match }
linkActErr            = 79; { No session, link not active }
sessActErr            = 80; { session activation failure }
limitsZeroErr         = 81; { mode limits zero }
unkCtloPerr           = 99; { unknown control operator error }

{ nodeOpErr }

procErr               = 1;
unkProcErr            = 2;
unkLinkNameErr        = 10;
unkALSNameErr         = 11;
unkPUNameErr          = 12;
activeReqErr          = 13;
dialErr               = 14;
unkLUNameErr          = 15;
unkMsgQueErr          = 16;
noMsgQueErr           = 17;
noMsgErr              = 18;
actPUErr              = 100;
actLUErr              = 110;
dctPUErr              = 120;
dctLUErr              = 130;
defMsgQueErr          = 140;
dspMsgQueErr          = 150;
dspMsgErr             = 190;

```

END.









## Appendix C



### MacAPPC Result Codes

This appendix gives a complete listing of all values for `appcHiResult` and `appcLoResult` and provides a detailed explanation of the meaning of each combination of codes.



---

---

## MacAPPC result codes

Major code	Name	Description
0	noErr	Function completed normally.
1	usageErr	Function aborted, usage error.
2	badComplErr	Function not completed.
3	stateErr	Function aborted, state error.
5	allocErr	Function aborted, allocation error.
7	progErr	Program error.
9	deallocErr	Deallocated.
10	ctlOpErr	Control operator error.
11	nodeOpErr	Node operator error.

---

### Major Code 00—noErr: Function completed normally

Major code	Name	Description
0	normalCompl	Function completed normally.
1	negotCompl	Function completed as negotiated.
3	dataAvail	MCTest or BCTest routine—data available.
4	ctlAvail	MCTest or BCTest routine—control information available.
6	badErr	Default; undefined.

---

### Major Code 01—usageErr: Function aborted, usage error

Major code	Name	Description
1	notAttachErr	Not attached. The transaction issued another routine before issuing a TPAttach routine. The program must attach before other routines may be called.
2	dupAttachErr	Duplicate attach. After a previous TPAttach routine and before a TPDetach routine, the transaction program attempted to issue the TPAttach routine again.
3	unkConvErr	Invalid conversation. The conversation specified in the cvConvID parameter is not a valid conversation.
4	badAttachErr	Routine not valid for current attach type specified on a previous TPAttach routine
5	tpcbErr	TPCB not set.
6	tpNameErr	TP name invalid.



7	lookupErr	Server lookup failed.
8	connectErr	Server connect failed.
9	noParamErr	Null structure passed. A null parameter structure pointer was passed.
10	srvrEntityErr	Server entity name required. The server entity name is required for TPAttach.
11	lclLUNameErr	LU name required. The LU name is required when the tpAttachType parameter is set to the kLUAttach or kWaitAttach constant.
12	attachTypeErr	Invalid attach type. Valid attach-type values for a transaction program are kCVCOAttach and kNOAttach.
13	unkLclLUNameErr	LU unknown. The specified LU is not configured.
14	availLUErr	LU unavailable. The LU specified in the attach is not activated, or has already reached the configured limit of attached transaction programs.
15	lclProgNameErr	Transaction program name unknown. The transaction program name specified in the request is not configured.
18	badConvErr	Invalid conversation for remote attach. The conversation specified in the TPAttach routine is invalid or cannot be remotely attached.
19	badWaitTimeErr	Invalid tpWaitTime parameter.
20	procSpaceErr	Attach rejected. Not enough space to support the transaction program.
21	procLimitErr	Attach rejected. The SNA server process limit has been reached.
25	cvcbErr	Conversation control block not set.
26	dataErr	Data pointer or size not set.
27	detachTypeErr	Invalid detach type. Valid detach-type values for a transaction program are kNormalDetach and kAbortDetach.
30	rmtLUNameErr	Remote LU name required.
31	modeNameErr	Mode name required.
32	rmtProgNameErr	Remote transaction program name required.
33	unkRmtLUNameErr	Remote LU unknown. The specified remote LU is not configured for the attached LU.
34	unkModeNameErr	Mode unknown. The specified mode is not configured between the remote LU and the attached LU.
35	convTypeErr	Invalid cvConvType parameter. Valid conversation-type values are kBasicConv and kMappedConv.
36	mapConvErr	Mapped conversations not available. Either the attached LU or the remote LU is not configured to support mapped conversations, or the transaction program name specified in the previous attach is not configured to support and authorize mapped conversations.



Minor code	Name	Description
37	basicConvErr	Basic conversations not available. The transaction program name specified in the previous attach is not configured to support and authorize basic conversations.
38	returnCtlErr	Invalid session allocation parameter. Valid <code>cvReturnCtl</code> parameters are <code>kWhenSessAllocReturn</code> , <code>kDelayAllocPermitReturn</code> , and <code>kImmediateReturn</code> .
39	delayAllocErr	No delayed session allocation. The attached LU is not configured to support delayed session allocation.
40	immedAllocErr	No immediate session allocation. The attached LU is not configured to support immediate session allocation.
41	syncTypeErr	Invalid sync-level parameter. Valid <code>cvSyncType</code> parameters are <code>kNoSync</code> and <code>kConfirmSync</code> .
42	badSyncErr	Requested sync-level not available. The requested sync-level support must be configured for the attached LU, the remote LU, the requested mode, and for the transaction program name specified in the previous attach request.
43	pipErr	No program initialization parameter data. PIP data support must be configured for the attached LU and the remote LU, and for the transaction program name specified in the previous attach request.
44	badPipErr	PIP data too large. Too many PIP parameters were passed in the PIP data, or the PIP data exceeds the maximum length.
45	badSecTypeErr	Invalid security-level parameter. Valid <code>cvSecType</code> values are <code>kNoSec</code> , <code>kSameSec</code> , and <code>kNameAndPswdSec</code> .
46	secTypeErr	Requested security level not available. The requested security level support is not configured. <code>kSameSec</code> must be specified for the attached LU and the remote LU. <code>kNameAndPswdSec</code> must be specified for the attached LU, the remote LU, and for the transaction program name specified in the previous attach request.
47	userNameFmtErr	Mode cannot specify the SNA-defined mode name <code>SNASVCMG</code> when using a mapped conversation.
48	userPswdFmtErr	Transaction program cannot specify an SNA service transaction program when using a mapped conversation.
49	pipBuffErr	Bad PIP buffer.
50	mapBuffErr	Bad mapped conversation buffer.
60	badSyncTypeErr	Sync-level conflict. A <code>BCCConfirm</code> or <code>BCCConfirmed</code> routine was attempted on a conversation that was allocated with the <code>cvSyncType</code> parameter set to the <code>kNoSync</code> constant.
61	sendReqErr	Write to server failed.
62	recvrSpErr	Read from server failed.
63	verbReqErr	Internal routine request failed.



70	deallocTypeErr	Invalid BC or MC deallocate type. Valid cvDeallocType values for BC routines are kSyncDealloc, kFlushDealloc, kAbendProgDealloc, kAbendSvcProgDealloc, kAbendTimerDealloc, kLocalDealloc, and kAbendDealloc. Valid cvDeallocType values for MC routines are kSyncDealloc, kFlushDealloc, kLocalDealloc, and kAbendDealloc.
80	prepToRcvTypeErr	Invalid prepare-to-receive type. Valid cvPrepToRcvType values are kFlushRcv, kConfirmRcv, and kSyncLevelRcv.
81	lockTypeErr	Invalid prepare-to-receive lock parameter. Valid cvLockType values are kShortLock and kLongLock. This parameter is significant when the cvPrepToRcvType parameter is set to the kSyncLevelRcv constant.
90	fillTypeErr	Invalid fill parameter. Valid cvFillType values for receive-and-wait and post-on-receipt are kBufferFill and kLLFill.
91	dataSizeErr	Invalid length. The cvDataSize parameter is too large.
100	badLLErr	Invalid logical length field. The data passed to BCSendData contains an invalid LL field of \$0000, \$8000, or \$8001.
101	badPSHdrLLErr	The data passed to the BCSendData routine contains a logical length field of \$0001, which indicates a PS header; PS headers are not supported.
110	errorTypeErr	Invalid send-error type. Valid cvErrorType values for transaction programs are kSvcError and kProgError.
110	badErrorTypeErr	An error type specified in a BCSendError or MCSendError routine is reserved.
112	logDataErr	Error log data not permitted. Either the attached LU or the remote LU is not configured to support error logging.
120	cvcbIndexErr	WAIT: An index less than 0 or greater than kMaxCVCB is invalid.
121	cvcbListErr	The cvCVCBList parameter passed to CVWait contained an invalid conversation ID.
122	badPostConvErr	WAIT: listed conversation not posting.
123	postConvErr	No conversations with posting active. If a list of conversations was passed to CVWait, none of the conversations had posting active. If no list was passed, then no conversations for the TP had posting active.
126	badWaitConvErr	WAIT: conversation posted not in CVCB list.
130	testTypeErr	TEST: invalid type.
131	postActErr	TEST: posting not active.
300	notAuthCOErr	TP not authorized to issue CNOS routines.
301	sessLimitErr	Limits requested in CNOS routine are invalid.
302	modeSVCMGRErr	CNOS routine issued before service manager mode (SNASVCMG) is initialized.



Minor code	Name	Description
310	zeroSessLimitErr	Session limit must be greater than zero.
311	respTypeErr	Invalid response type parameter.
312	sessSumErr	Sum of the minimum first speaker sessions and the minimum bidder sessions cannot exceed session limit.
313	deactTypeErr	Illegal deactivate type specified for a CODEactivateSession routine.
314	drainTgtErr	Illegal drain target specified for a COREsetSessionLimit routine.
315	drainSrcErr	Illegal drain source specified for a COREsetSessionLimit routine.
316	forceRstErr	Illegal force reset specified for a COREsetSessionLimit routine.
320	badSessIDErr	Invalid session identifier.
321	nextSessErr	No next session found.
400	nameErr	For a node operator routine, a string parameter was too long.
401	lineErr	For a node operator routine, a line name is required.
402	luErr	For a node operator routine, a LU name is required.
403	puErr	For a node operator routine, a PU name is required.
404	dialTypeErr	For a node operator routine, an invalid noDialType parameter was specified. Valid noDialType values are kConnectDial, kDisconnectDial, kDialInOnDial, and kDialInOffDial.
405	queueNameErr	For a node operator routine, an invalid queue name was specified.
411	sevTypeErr	For a node operator routine, an invalid severity was specified.
501	invLclLUNameErr	Invalid value for the coLclLUName parameter of a CODEfineLocalLU routine.
502	invRmtLUNameErr	The value specified for the coDefLUMaxSess parameter of the CODEfineLocalLU routine is less than the sum of the currently defined LU-mode session limits.
503	invModeNameErr	For a node operator definition or control operator definition, an invalid mode was specified.
504	alsNameErr	For a node operator definition or control operator definition, an invalid ALS name was specified.
505	sendPacingErr	For a node operator definition or control operator definition, an invalid send pacing window was specified.
506	recvPacingErr	For a node operator definition or control operator definition, an invalid receive pacing window was specified.
507	maxRUHiBoundErr	For a node operator definition or control operator definition, an invalid maximum RU upper bound was specified.
508	maxRULoBoundErr	For a node operator definition or control operator definition, an invalid maximum RU lower bound was specified.



509	invSyncTypeErr	For a node operator definition or control operator definition, an invalid sync-level option was specified.
510	reinitTypeErr	For a node operator definition or control operator definition, an invalid reinit option was specified.
511	sessCryptErr	For a node operator definition or control operator definition, an invalid session-level cryptography option was specified.
512	defMaxSessErr	For a node operator definition or control operator definition, an invalid maximum number of sessions was specified.
513	defMinFirstSpkrsErr	For a node operator definition or control operator definition, an invalid minimum number of first speakers was specified.
514	defPBFirstSpkrsErr	For a node operator definition or control operator definition, an invalid minimum number of first prebound speakers was specified.
515	queueBINDsErr	For a node operator definition or control operator definition, an invalid bind queue option was specified.
516	blankModeErr	For a node operator definition or control operator definition, an invalid blank mode option was specified.
517	netNameOpErr	For a node operator definition or control operator definition, an invalid network name operation was specified.
518	netQualOpErr	For a node operator definition or control operator definition, an invalid network qualifier operation was specified.
519	netNameErr	For a node operator definition or control operator definition, an invalid network name was specified.
520	netQualErr	For a node operator definition or control operator definition, an invalid network qualifier was specified.
521	queueINITsErr	For a node operator definition or control operator definition, an invalid queue init option was specified.
522	parSessErr	For a node operator definition or control operator definition, an invalid parallel session option was specified.
523	cnosALSNameErr	For a node operator definition or control operator definition, an invalid CNOS ALS name was specified.
524	luPswdErr	For a node operator definition or control operator definition, an invalid LU password was specified.
525	secAccErr	For a node operator definition or control operator definition, an invalid security-accepted option was specified.
526	luPswdOpErr	For a node operator definition or control operator definition, an invalid password operation was specified.
527	defLUMaxSessErr	For a node operator definition or control operator definition, an invalid LU session limits was specified.
528	convSecErr	For a node operator definition or control operator definition, an invalid conversation security was specified.



Minor code	Name	Description
529	secOpErr	For a node operator definition or control operator definition, an invalid security operation was specified.
530	userNameErr	For a node operator definition or control operator definition, an invalid user ID was specified.
531	userPswdErr	For a node operator definition or control operator definition, an invalid password was specified.
532	userProfErr	For a node operator definition or control operator definition, an invalid profile was specified.
533	waitTimeErr	For a node operator definition or control operator definition, an invalid wait period was specified.
534	maxTPerr	For a node operator definition or control operator definition, an invalid maximum number of TPs was specified.
535	luIDerr	For a node operator definition or control operator definition, an invalid LU ID number was specified.
536	tpNameDefErr	For a node operator definition or control operator definition, an invalid transaction program name was specified.
537	luActiveErr	For a node operator definition or control operator definition, an invalid status was specified.
538	invConvTypeErr	For a node operator definition or control operator definition, an invalid conversation type was specified.
539	secReqErr	For a node operator definition or control operator definition, an invalid security-required option was specified.
540	pipReqErr	For a node operator definition or control operator definition, an invalid PIP option was specified.
541	pipCountErr	For a node operator definition or control operator definition, an invalid PIP number was specified.
542	pipCheckErr	For a node operator definition or control operator definition, an invalid PIP check was specified.
543	dataMapErr	For a node operator definition or control operator definition, an invalid data mapping option was specified.
544	fmhDataSuppErr	For a node operator definition or control operator definition, an invalid FMH was specified.
545	privTypeErr	For a node operator definition or control operator definition, an invalid privilege was specified.
546	luwSuppErr	For a node operator definition or control operator definition, an invalid LUW indicator was specified.
550	unkLUerr	For a node operator definition or control operator definition, the Local LU is unknown.
551	unkRmtErr	For a node operator definition or control operator definition, the Remote LU is unknown.
552	unkALSErr	For a node operator definition or control operator definition, the ALS is unknown.
553	defSpaceErr	For a node operator definition or control operator definition, not enough server memory space for definition.



554	unkRmtPUErr	For a node operator definition or control operator definition, the control point is unknown.
555	unkLineErr	For a node operator definition or control operator definition, the line is unknown.
556	unkLSCBErr	For a node operator definition or control operator definition, LSCB not found.
557	lineActiveErr	For a node operator definition or control operator definition, the associated line is not inactive.
558	unkALCBErr	For a node operator definition or control operator definition, ALCB not found.
560	modeParamErr	For a control operator definition routine, an invalid parameter was specified.
561	maxRUBoundErr	For a control operator definition routine, the lower bound is greater than the upper bound.
562	reinitErr	For a control operator definition routine, there was a single session reinit error.
563	lclStationErr	For a control operator definition routine, the station was local and the remote LU was not local.
564	minFirstSpkrsErr	For a control operator definition routine, the minimum number of first speakers was greater than the maximum number of sessions.
565	pbFirstSpkrsErr	For a control operator definition routine, the minimum number of prebound first speakers was greater than the minimum number of first speakers.
566	badMaxSessErr	For a control operator definition routine, an invalid number of maximum sessions was specified.
567	badBlankModeErr	For a control operator definition routine, the blank mode already exists.
570	cnosALSErr	For a CODEfineRemoteLU routine, there was a CNOS ALS and parallel session support error.
571	parSessMaxSessErr	For a CODEfineRemoteLU routine, there was a parallel session and maximum session error.
572	lclALSBadNameErr	For a CODEfineRemoteLU routine, the ALS name is local and the remote name is not the same as the local name.
573	rmtALSBadNameErr	For a CODEfineRemoteLU routine, the ALS name is not local and the remote name is the same as the local name.
574	parSessReinitErr	For a CODEfineRemoteLU routine, an invalid reinit option for parallel sessions was specified.
575	unkCNOSALSErr	For a CODEfineRemoteLU routine, CNOS ALS unknown.
576	badNetNameErr	For a CODEfineRemoteLU routine, null net name already exists.
577	parSessPUTypeErr	For a CODEfineRemoteLU routine, there was a parallel session and PU-type error.
580	lclPUErr	For a CODEfineRemoteLU routine, a local PU structure does not exist.



Minor code	Name	Description
581	maxSessLimitErr	For a CODEfineRemoteLU routine, session limits were greater than the defined LU session limits.
582	initNetNameErr	For a CODEfineRemoteLU routine, the initial net name specified was invalid.
583	unkLUSecErr	For a CODEfineRemoteLU routine, the security parameters to be deleted were not found.
584	badLUIDErr	For a CODEfineRemoteLU routine, the LU ID was already specified.
585	initLUIDErr	For a CODEfineRemoteLU routine, the LU ID must be specified.
586	luIDUpdateErr	For a CODEfineRemoteLU routine, the LU ID can't be updated.
590	badSecReqErr	For a CODEfineRemoteLU routine, the security-access parameters did not correspond to the required security.
591	unkTPSecErr	For a CODEfineRemoteLU routine, security parameters to be deleted were not found.
595	accessTypeErr	For a node operator definition or control operator definition, an invalid access type was specified.
596	lineNumErr	For a node operator definition or control operator definition, an invalid line number was specified.
600	eaErr	For a node operator definition or control operator definition, a resource was not available.
601	exchIDErr	For a node operator definition or control operator definition, an invalid exchange ID was specified.
602	invDevNameErr	For a node operator definition or control operator definition, an invalid master device name was specified.
603	monTimerErr	For a node operator definition or control operator definition, an invalid monitor timer value was specified.
604	nodeMsgsErr	For a node operator definition or control operator definition, an invalid NOOP message was specified.
605	logMsgsErr	For a node operator definition or control operator definition, an invalid message was specified.
606	debugMsgsErr	For a node operator definition or control operator definition, an invalid debug message was specified.
607	cpNameErr	For a node operator definition or control operator definition, an invalid PU name was specified.
608	cpuIDErr	For a node operator definition or control operator definition, an invalid CPU ID was specified.
609	lineNameErr	For a node operator definition or control operator definition, an invalid line name was specified.
610	lineTypeErr	For a node operator definition or control operator definition, an invalid line type was specified.
612	sdlcRoleErr	For a node operator definition or control operator definition, an invalid SDLC role was specified.



613	connTypeErr	For a node operator definition or control operator definition, an invalid connection type was specified.
614	nrziTypeErr	For a node operator definition or control operator definition, an invalid NRZI type was specified.
615	duplexTypeErr	For a node operator definition or control operator definition, an invalid duplex value was specified.
616	maxBTUErr	For a node operator definition or control operator definition, an invalid maximum BTU was specified.
617	maxRetrysErr	For a node operator definition or control operator definition, an invalid maximum retries value was specified.
618	idleTimeErr	For a node operator definition or control operator definition, an invalid idle time value was specified.
619	npRecvTimeErr	For a node operator definition or control operator definition, an invalid nonproductive time was specified.
620	maxIFramesErr	For a node operator definition or control operator definition, an invalid maximum number of I-frames was specified.
621	lineSpeedErr	For a node operator definition or control operator definition, an invalid line speed was specified.
622	alsAddrErr	For a node operator definition or control operator definition, an invalid station address was specified.
623	phoneNumberErr	For a node operator definition or control operator definition, an invalid phone number was specified.
624	maxSOTerr	For a node operator definition or control operator definition, a value was specified that exceeded the maximum SOT.
626	badSDLCRoleErr	For a node operator definition or control operator definition, a line value was specified that was multipoint and the role was not primary.
627	badLineTypeErr	For a node operator definition or control operator definition, a station was already defined for a leased line.
628	badLineErr	For a node operator definition or control operator definition, a line already exists.
630	unkLineNameErr	For a node operator definition or control operator definition, a line was not specified.
631	unkPUErr	For a node operator definition or control operator definition, a CP was not specified.
632	badStationErr	For a node operator definition or control operator definition, a station already exists.
634	badLineNameErr	For a node operator display or control operator display routine, an invalid line name was specified.
635	endLineListErr	For a node operator display or control operator display routine, the end of the line list was reached.
636	noLineListErr	For a node operator display or control operator display routine, no lines were defined.



Minor code	Name	Description
637	badPUNameErr	For a NODefineCP routine, an invalid CP name was specified.
638	endPUListErr	For a NODefineCP routine, the end of the CP list was reached.
639	noPUListErr	For a NODefineCP routine, no CPs were defined.
640	unkModeNameDelErr	For a node operator definition or control operator definition, a mode to be deleted was unknown.
640	unkModeNameDelErr	For a node operator definition or control operator definition, a mode to be deleted was unknown.
642	objInUseErr	For a node operator definition or control operator definition, an object to be deleted is in use.
640	unkModeNameDelErr	For a node operator definition or control operator definition, a mode to be deleted was unknown.
640	unkModeNameDelErr	For a node operator definition or control operator definition, a mode to be deleted was unknown.
648	delNameErr	For a node operator definition or control operator definition, no parameters were specified in a delete request.
650	xidOrCPIDErr	For a NODefineCP routine, neither the XID or the CPUID were specified.
651	nodeIDErr	For a NODefineCP routine, both the XID and the CPUID were specified.
660	badUserNameErr	For a CODisplayLocalLU routine, the user ID was invalid.
661	endSecListErr	For a CODisplayLocalLU routine, the end of the security list was reached.
662	unkUserNameErr	For a CODisplayLocalLU routine, no user ID was found.
663	badUserProfErr	For a CODisplayLocalLU routine, the profile was invalid.
664	badLUNameErr	For a CODisplayLocalLU routine, the LU name was invalid.
665	endLUListErr	For a CODisplayLocalLU routine, the end of the LU list was reached.
666	luNameErr	For a CODisplayLocalLU routine, no LUs were defined.
667	badRmtLUNameErr	For a CODisplayRemoteLU routine, the remote LU name was invalid.
668	endRmtLUListErr	For a CODisplayRemoteLU routine, the end of the remote LU list was reached.
669	noRmtLUListErr	For a CODisplayRemoteLU routine, no remote LUs were defined.
670	badModeNameErr	For a CODisplayMode routine, an invalid mode name was specified.
671	endModeListErr	For a CODisplayMode routine, the end of the mode list was reached.
672	noModeListErr	For a CODisplayMode routine, no mode was defined.
673	badTPNameErr	For a CODisplayTP routine, an invalid TP name was specified.
674	endTPListErr	For a CODisplayTP routine, the end of the TP list was reached.



675	noTPListErr	For a CODisplayTP routine, no TPs were defined.
676	noNetNameListErr	For a CODisplayTP routine, no network name was specified.
840	srvrSpaceErr	The server is out of request space.
850	srvrNotActiveErr	The MacAPPC server is not active.
860	srvrCrashedErr	The MacAPPC server is no longer active. The transaction program has been detached.
870	timeOutNoRecErr	A time-out request was not recovered.
880	timeOutRecErr	A time-out request was recovered.
890	timeOutQueFullErr	A time out occurred because the message queue was full.
900	logicErr	An internal logic error occurred while processing the routine.

---

### Major Code 02—badComplEr: Function aborted, bad completion

Minor code	Name	Description
1	rcvImmErr	An MReceiveImmediate or BReceiveImmediate routine was unsuccessful.
2	noDataErr	No data or information was available to post.
3	noReqToSendErr	No request was available to send.

---

### Major Code 03—stateErr: Function aborted, state error

Minor code	Name	Description
1	convStateErr	Request is illegal in the current conversation state.
2	logicalRecErr	Request is illegal because the current logical record has not been completed.
3	waitConvStateErr	A conversation specified in the wait request is not in receive state.

---

### Major Code 05—allocErr: Function aborted, allocation error

Minor code	Name	Description
1	tpNoRetryErr	Transaction program could not be started on the remote system because of a lack of resources which is not temporary. Retry is not suggested.
2	tpRetryErr	Transaction program could not be started on the remote system because of a temporary resource shortage. Retry is suggested.
3	convTypeMatchErr	The remote transaction program does not support the requested conversation type.



Minor code	Name	Description
4	pipSuppErr	Program initialization parameter data not supported by the remote.
5	pipSpecErr	Program initialization parameter data was specified incorrectly.
6	secInfoErr	Security information was not specified correctly.
7	syncTypeSuppErr	Remote program doesn't support requested sync level.
8	unkProgNameErr	Transaction program requested was not recognized at the remote LU.
9	allocNoRetryErr	An allocation failure occurred; no retry was attempted.
10	allocRetryErr	An allocation failure occurred; a retry was attempted.
11	immSessErr	A session was not immediately available.
12	rsrcFailErr	A local resource allocation failure occurred.
13	badAllocErr	An allocation error occurred.
16	allocSubErr	An allocation error occurred (subcoded).
22	allocGenErr	A generic allocation failure occurred.
99	unkAllocErr	An unknown allocation error occurred.

---

### Major Code 07—progErr: Program error

Minor code	Name	Description
1	progNoTruncErr	Partner has reported a program error; the current logical record was not truncated.
2	progLLTruncErr	Partner has reported a program error; the current logical record was truncated.
3	progPurgingErr	Partner has reported a program error; the current logical record may have been purged.
11	svcNoTruncErr	Service transaction program has reported an error; the current logical record was not truncated.
12	svcLLTruncErr	Service transaction program has reported an error; the current logical record was truncated.
13	svcPurgingErr	Service transaction program has reported a program error; the current logical record may have been purged.
20	fmhSuppErr	FMH data is not supported.
21	mapSuppErr	Mapping is not supported.
22	mapNameErr	The specified map procedure was not found.
23	mapProcErr	The map procedure failed.
24	dupMapNameErr	A duplicate map name was specified.



---

## Major Code 09—deallocErr: Deallocated

Minor code	Name	Description
0	normDeallocErr	Normal deallocation. Partner has deallocated the conversation.
1	abendProgErr	Abnormal deallocation of a transaction program. Partner has terminated the conversation abnormally.
2	abendSvcErr	Abnormal deallocation of a service transaction program. A service transaction program has terminated the conversation abnormally.
3	abendTimerErr	Abnormal deallocation—time out has occurred.
4	rsrcErr	A resource failure occurred.
5	abendErr	Session has failed.

---

## Major Code 10—ctlOpErr: Control operator error

Minor code	Name	Description
63	limitsNotZeroErr	Limits are not zero.
64	limitsTooBigErr	Requested limits exceed configuration.
65	limitsSumTooBigErr	Sum of the minimum first speaker and bidder exceeds the requested maximum.
66	badSNASVCMGLimitsErr	Invalid SNASVCMG limits; must be maximum of two, one bidder, one first speaker.
67	modeSNASVCMGInitErr	SNASVCMG mode not initialized.
68	limitsClosedErr	Mode limits are closed.
69	noCNOSerr	Change-session limit not valid for this mode.
70	userModesErr	SNASVCMG mode can't be reset because user modes are still open.
71	raceCNOSerr	CNOS race at remote—they won.
72	unkModeErr	Partner doesn't recognize mode.
73	localCNOSerr	CNOS is in process locally.
74	allocCNOSerr	CNOS allocation error.
75	rsrcCNOSerr	CNOS resource failure. The SNASVCMG session with the partner LU either could not be started, or failed
76	noSessSpaceErr	Not enough space for session.
77	rmtLUActErr	The partner LU was not active.
78	modeParmErr	Mode configurations don't match.
79	linkActErr	No session, link not active.
80	sessActErr	Session failed to activate.
81	limitsZeroErr	The mode limits were zero.
99	unkCtlOpErr	An unknown control operator error occurred.



---

## Major Code 11—nodeOpErr: Node operator error

Minor code	Name	Description
1	procErr	Node operator routine failure.
2	unkProcErr	Node operator routine not recognized.
10	unkLinkNameErr	Link name not recognized.
11	unkALSNameErr	Adjacent link station not recognized.
12	unkPUNameErr	Control point not recognized.
13	activeReqErr	Earlier request still active.
14	dialErr	Dial-in or dial-out required.
15	unkLUNameErr	Logical unit name not recognized.
16	unkMsgQueErr	Message queue name not recognized.
17	noMsgQueErr	Message queue not enabled.
18	noMsgErr	No message in message queue.
100	actPUErr	NOActivateNode routine failure.
110	actLUErr	NOActivateLU routine failure.
120	dctPUErr	NODEactivateNode routine failure.
130	dctLUErr	NODEactivateLU routine failure.
140	defMsgQueErr	NODefineMessageQueue routine failure.
150	dspMsgQueErr	NODisplayMessageQueue routine failure.
190	dspMsgErr	NODisplayMessage routine failure.





## Appendix D



### MacAPPC Routine Mapping

This appendix gives a complete listing of the mapping between each LU 6.2 verb and its corresponding MacAPPC routine. *N/A* indicates that there is no corresponding LU 6.2 verb.



---

---

## Conversation routine mapping

### LU 6.2 verb

### MacAPPC routine

MC_ALLOCATE	MCAAllocate
MC_CONFIRM	MCConfirm
MC_CONFIRMED	MCConfirmed
MC_DEALLOCATE	MCDeallocate
MC_FLUSH	MCFlush
MC_GET_ATTRIBUTES	MCGetAttributes
MC_POST_ON_RECEIPT	MCPostOnReceipt
MC_PREPARE_TO_RECEIVE	MCPrepareToReceive
MC_RECEIVE_AND_WAIT	MCReceiveAndWait
MC_RECEIVE_IMMEDIATE	MCReceiveImmediate
MC_REQUEST_TO_SEND	MCRequestToSend
MC_SEND_DATA	MCSendData
MMC_SEND_ERROR	MCSendError
MC_TEST	MCTest
BACKOUT	CVBackout
GET_TYPE	CVGetType
SYNCP	CVSyncPoint
WAIT	CVWait
ALLOCATE	BCAllocate
CONFIRM	BCConfirm
CONFORMED	BCConfirmed
DEALLOCATE	BCDeallocate
FLUSH	BCFlush
GET_ATTRIBUTES	BCGetAttributes
POST_ON_RECEIPT	BCPostOnReceipt
PREPARE_TO_RECEIVE	BCPrepareToReceive
RECEIVE_AND_WAIT	BCReceiveAndWait
RECEIVE_IMMEDIATE	BCReceiveImmediate
REQUEST_TO_SEND	BCRequestToSend
SEND_DATA	BCSendData
SEND_ERROR	BCSendError
TEST	BCTest



---

---

## Control operator routine mapping

LU 6.2 verb	MacAPPC routine
CHANGE_SESSION_LIMIT	COChangeSessionLimit
INITIALIZE_SESSION_LIMIT	COInitializeSessionLimit
RESET_SESSION_LIMIT	COResetSessionLimit
PROCESS_SESSION_LIMIT	COProcessSessionLimit
ACTIVATE_SESSION	COActivateSession
DEACTIVATE_SESSION	CODEactivateSession
DEFINE_LOCAL_LU	CODefineLocalLU
DEFINE_REMOTE_LU	CODefineRemoteLU
DEFINE_MODE	CODefineMode
DEFINE_TP	CODefineTP
DISPLAY_LOCAL_LU	CODisplayLocalLU
DISPLAY_REMOTE_LU	CODisplayRemoteLU
DISPLAY_MODE	CODisplayMode
DISPLAY_TP	CODisplayTP
DELETE	CODElete

---

---

## Node operator routine mapping

LU 6.2 verb	MacAPPC routine
N/A	NOActivateLine
N/A	NOActivateLU
N/A	NOActivateNode
N/A	NOActivateStation
N/A	NODEactivateLine
N/A	NODEactivateLU
N/A	NODEactivateNode
N/A	NODEactivateStation
N/A	NODEfineMessageQueue
N/A	NODisplayMessage
N/A	NODisplayMessageQueue
N/A	NODEfineCP
N/A	NODEfineLine
N/A	NODEfineNode



**LU 6.2 verb****MacAPPC routine**

N/A	NODefineStation
N/A	NODelete
N/A	NODisplayCP
N/A	NODisplayLine
N/A	NODisplayNode
N/A	NODisplayStation

---

---

**Transaction program routine mapping****LU 6.2 verb****MacAPPC routine**

N/A	TPAttach
N/A	TPDetach
N/A	TPAsciiToEbcDic
N/A	TPEbcDicToAscii





## Appendix E



### MacAPPC Conversation Parameter Mapping

This appendix gives a complete listing of the mapping between the parameters for each LU 6.2 conversation verb parameter and its corresponding MacAPPC conversation routine parameter.



---

---

## MC\_ALLOCATE is MCAAllocate

LU 6.2 parameter	MacAPPC parameter
LU_NAME	cvRmtLUName
MODE_NAME	cvModeName
TPN	cvRmtProgName
RETURN_CONTROL	cvReturnCtl
SYNC_LEVEL	cvSyncType
SECURITY	cvSecType
USER_ID	cvUserName
PASSWORD	cvUserPswd
PROFILE	cvUserProf
PIP	cvPIPUsed
	cvPIPPtr[]
	cvPIPSize[]
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult
	appcLoResult

---

---

## MC\_CONFIRM is MCCConfirm

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult
	appcLoResult
REQUEST_TO_SEND_RECEIVED	cvReqToSendRcvd

---

---

## MC\_CONFIRMED is MCCConfirmed

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult
	appcLoResult



---

---

## MC\_DEALLOCATE is MCDeallocate

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
TYPE	cvDeallocType
RETURN_CODE	appcHiResult
	appcLoResult

---

---

## MC\_FLUSH is MCFlush

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult
	appcLoResultt

---

---

## MC\_GET\_ATTRIBUTES is MCGetAttributes

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
OWN_FULLY_QUALIFIED_LU_NAME	cvFullLclLUName
PARTNER_LU_NAME	cvRmtLUName
PARTNER_FULLY_QUALIFIED_LU_NAME	cvFullRmtLUName
MODE_NAME	cvModeName
SYNC_LEVEL	cvSyncType
SECURITY_USER_ID	cvUserName
SECURITY_PROFILE	cvUserProf
LUW_IDENTIFIER	cvLUWID
CONVERSATION_CORRELATOR	cvLUWCorr
RETURN_CODE	appcHiResult
	appcLoResult



---

---

## MC\_POST\_ON\_RECEIPT is MCTPostOnReceipt

### LU 6.2 parameter

RESOURCE  
LENGTH  
RETURN\_CODE

### MacAPPC parameter

cvCVCBPtr  
cvDataSize  
appcHiResult  
appcLoResult

---

---

## MC\_PREPARE\_TO\_RECEIVE is MCTPrepareToReceive

### LU 6.2 parameter

RESOURCE  
TYPE  
LOCKS  
RETURN\_CODE

### MacAPPC parameter

cvCVCBPtr  
cvPrepToRcvType  
cvLockType  
appcHiResult  
appcLoResult

---

---

## MC\_RECEIVE\_AND\_WAIT is MCTReceiveAndWait

### LU 6.2 parameter

RESOURCE  
LENGTH  
RETURN\_CODE  
  
REQUEST\_TO\_SEND\_RECEIVED  
DATA  
WHAT\_RECEIVED  
MAP\_NAME

### MacAPPC parameter

cvCVCBPtr  
cvDataSize  
appcHiResult  
appcLoResult  
cvReqToSendRcvd  
cvDataPtr  
cvWhatRcvd  
cvMapName



---

---

## MC\_RECEIVE\_IMMEDIATE is MReceiveImmediate

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE	cvCVCBPtr
LENGTH	cvDataSize
RETURN_CODE	appcHiResult
	appcLoResult
REQUEST_TO_SEND_RECEIVED	cvReqToSendRcvd
DATA	cvDataPtr
WHAT_RECEIVED	cvWhatRcvd
MAP_NAME	cvMapName

---

---

## MC\_REQUEST\_TO\_SEND is MRequestToSend

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult
	appcLoResult

---

---

## MC\_SEND\_DATA is MSendData

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE	cvCVCBPtr
DATA	cvDataPtr
LENGTH	cvDataSize
MAP_NAME	cvMapName
FMH_DATA	cvFMHdrs
RETURN_CODE	appcHiResult
	appcLoResult
REQUEST_TO_SEND_RECEIVED	cvReqToSendRcvd



---

---

## MC\_SEND\_ERROR is MCSendError

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult appcLoResult
REQUEST_TO_SEND_RECEIVED	cvReqToSendRcvd

---

---

## MC\_TEST is MCTest

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
TEST	cvTestType
RETURN_CODE	appcHiResult appcLoResult

---

---

## BACKOUT is CVBackout

LU 6.2 parameter	MacAPPC parameter
RETURN_CODE	appcHiResult appcLoResult

---

---

## GET\_TYPE is CVGetType

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
TYPE	cvConvType
RETURN_CODE	appcHiResult appcLoResult



---

---

## SYNCPt is CVSyncPoint

LU 6.2 parameter	MacAPPC parameter
RETURN_CODE	appcHiResult appcLoResult
REQUEST_TO_SEND_RECEIVED	cvReqToSendRcvd

---

---

## WAIT is CVWait

LU 6.2 parameter	MacAPPC parameter
RESOURCE_LIST	cvCVCBList [] cvCVCBIndex
RETURN_CODE	appcHiResult appcLoResult
RESOURCE_POSTED	cvCVCBPtr

---

---

## ALLOCATE is BCAllocate

LU 6.2 parameter	MacAPPC parameter
LU_NAME	cvRmtLUName
MODE_NAME	cvModeName
TPN	cvRmtProgName
TYPE	cvConvType
RETURN_CONTROL	cvReturnCtl
SYNC_LEVEL	cvSyncType
SECURITY	cvSecType
USER_ID	cvUserName
PASSWORD	cvUserPswd
PROFILE	cvUserProf
PIP	cvPIPUsed cvPIPPtr [] cvPIPSize []
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult appcLoResult



---

---

## CONFIRM is BCCConfirm

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult appcLoResult
REQUEST_TO_SEND_RECEIVED	cvReqToSendRcvd

---

---

## CONFIRMED is BCConfirmed

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult appcLoResult

---

---

## DEALLOCATE is BCDeallocate

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
TYPE	cvDeallocType
LOG_DATA	cvDataPtr cvDataSize
RETURN_CODE	appcHiResult appcLoResult

---

---

## FLUSH is BCFlush

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult appcLoResult



---

---

## GET\_ATTRIBUTES is BCGetAttributes

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE	cvCVCBPtr
OWN_FULLY_QUALIFIED_LU_NAME	cvFullLclLUName
PARTNER_LU_NAME	cvRmtLUName
PARTNER_FULLY_QUALIFIED_LU_NAME	cvFullRmtLUName
MODE_NAME	cvModeName
SYNC_LEVEL	cvSyncType
SECURITY_USER_ID	cvUserName
SECURITY_PROFILE	cvUserProf
LUW_IDENTIFIER	cvLUWID
CONVERSATION_CORRELATOR	cvLUWCorr
RETURN_CODE	appcHiResult
	appcLoResult

---

---

## POST\_ON\_RECEIPT is BCPostOnReceipt

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE	cvCVCBPtr
FILL	cvFillType
LENGTH	cvDataSize
RETURN_CODE	appcHiResult
	appcLoResult

---

---

## PREPARE\_TO\_RECEIVE is BCPrepareToReceive

### LU 6.2 Parameter

### MacAPPC parameter

RESOURCE	cvCVCBPtr
TYPE	cvPrepToRcvType
LOCKS	cvLockType
RETURN_CODE	appcHiResult
	appcLoResult



---

---

## RECEIVE\_AND\_WAIT is BCRceiveAndWait

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
FILL	cvFillType
LENGTH	cvDataSize
RETURN_CODE	appcHiResult appcLoResult
REQUEST_TO_SEND_RECEIVED	cvReqToSendRcvd
DATA	cvDataPtr
WHAT_RECEIVED	cvWhatRcvd

---

---

## RECEIVE\_IMMEDIATE is BCRceiveImmediate

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
FILL	cvFillType
LENGTH	cvDataSize
RETURN_CODE	appcHiResult appcLoResult
REQUEST_TO_SEND_RECEIVED	cvReqToSendRcvd
DATA	cvDataPtr
WHAT_RECEIVED	cvWhatRcvd

---

---

## REQUEST\_TO\_SEND is BCRequestToSend

LU 6.2 parameter	MacAPPC parameter
RESOURCE	cvCVCBPtr
RETURN_CODE	appcHiResult appcLoResult



---

---

## SEND\_DATA is BCSendData

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE

cvCVCBPtr

DATA

cvDataPtr

LENGTH

cvDataSize

RETURN\_CODE

appcHiResult

appcLoResult

REQUEST\_TO\_SEND\_RECEIVED

cvReqToSendRcvd

---

---

## SEND\_ERROR is BCSendError

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE

cvCVCBPtr

TYPE

cvErrorType

LOG\_DATA

cvDataPtr

cvDataSize

RETURN\_CODE

appcHiResult

appcLoResult

REQUEST\_TO\_SEND\_RECEIVED

cvReqToSendRcvd

---

---

## TEST is BCTest

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE

cvCVCBPtr

TEST

cvTestType

RETURN\_CODE

appcHiResult

appcLoResult









## Appendix F



### MacAPPC Control Operator Parameter Mapping

This appendix gives a complete listing of the mapping between the parameters for each LU 6.2 control operator verb parameter and its corresponding MacAPPC control operator routine parameter. *N/A* indicates that there is no corresponding parameter.



---

---

## CHANGE\_SESSION\_LIMIT is COChangeSessionLimit

### LU 6.2 parameter

LU\_NAME  
MODE\_NAME  
LU\_MODE\_SESSION\_LIMIT  
MIN\_CONWINNERS\_SOURCE  
MIN\_CONWINNERS\_TARGET  
RESPONSIBLE  
RETURN\_CODE

### MacAPPC parameter

coRmtLUName  
coModeName  
coCurMaxSess  
coCurMinFirstSpkrs  
coCurMinBdrs  
coRespType  
appcHiResult  
appcLoResult

---

---

## INITIALIZE\_SESSION\_LIMIT is COInitializeSessionLimit

### LU 6.2 parameter

LU\_NAME  
MODE\_NAME  
LU\_MODE\_SESSION\_LIMIT  
MIN\_CONWINNERS\_SOURCE  
MIN\_CONWINNERS\_TARGET  
RETURN\_CODE

### MacAPPC parameter

coRmtLUName  
coModeName  
coCurMaxSess  
coCurMinFirstSpkrs  
coCurMinBdrs  
appcHiResult  
appcLoResult

---

---

## RESET\_SESSION\_LIMIT is COREsetSessionLimit

### LU 6.2 parameter

LU\_NAME  
MODE\_NAME  
RESPONSIBLE  
DRAIN\_SOURCE  
DRAIN\_TARGET  
FORCE  
RETURN\_CODE

### MacAPPC parameter

coRmtLUName  
coModeName  
coRespType  
coDrainSrc  
coDrainTgt  
coForceRst  
appcHiResult  
appcLoResult



---

---

## PROCESS\_SESSION\_LIMIT is COProcessSessionLimit

### LU 6.2 parameter

### MacAPPC parameter

RESOURCE

coConvID

LU\_NAME

coRmtLUName

MODE\_NAME

coModeName

RETURN\_CODE

appcHiResult

appcLoResult

---

---

## ACTIVATE\_SESSION is COActivateSession

### LU 6.2 parameter

### MacAPPC parameter

LU\_NAME

coRmtLUName

MODE\_NAME

coModeName

RETURN\_CODE

appcHiResult

appcLoResult

---

---

## DEACTIVATE\_SESSION is CODEactivateSession

### LU 6.2 parameter

### MacAPPC parameter

SESSION\_ID

coSessID

TYPE

coDeactType

RETURN\_CODE

appcHiResult

appcLoResult

---

---

## DEFINE\_LOCAL\_LU is CODEfineLocalLU

### LU 6.2 parameter

### MacAPPC parameter

FULLY\_QUALIFIED\_LU\_NAME

coLclLUName

coNetNameOp

coNetName

coNetQualOp

coNetQual

LU\_SESSION\_LIMIT

coDefLUMaxSess



**LU 6.2 parameter****MacAPPC parameter**

SECURITY

coConvSecType

coSecOp

coUserName

coUserPswd

coUserProf

MAP\_NAME

N/A

RETURN\_CODE

appcHiResult

appcLoResult

---

---

**DEFINE\_REMOTE\_LU is CDefineRemoteLU****LU 6.2 parameter****MacAPPC parameter**

FULLY\_QUALIFIED\_LU\_NAME

coNetNameOp

coNetName

coNetQualOp

coNetQual

LOCALLY\_KNOWN\_LU\_NAME

coRmtLUName

UNINTERPRETED\_LU\_NAME

N/A

INITIATE\_TYPE

coQueueINITs

PARALLEL\_SESSION\_SUPPORT

coParSess

CNOS\_SUPPORT

coCNOSALSName

LU\_LU\_PASSWORD

coLUPswdOp

coLUPswd

SECURITY\_ACCEPTANCE

coLclSecAcc

RETURN\_CODE

appcHiResult

appcLoResult

---

---

**DEFINE\_MODE is CDefineMode****LU 6.2 parameter****MacAPPC parameter**

FULLY\_QUALIFIED\_LU\_NAME

coRmtLUName

MODE\_NAME

coModeName

SEND\_PACING\_WINDOW

coSendPacing

RECEIVE\_PACING\_WINDOW

coRcvPacing

SEND\_MAX\_RU\_SIZE\_LOWER\_BOUND

coMaxRUloBound

SEND\_MAX\_RU\_SIZE\_UPPER\_BOUND

coMaxRUHiBound



RECEIVE_MAX_RU_SIZE_LOWER_BOUND	coMaxRULoBound
RECEIVE_MAX_RU_SIZE_UPPER_BOUND	coMaxRUHiBound
SYNC_LEVEL_SUPPORT	coSyncType
SINGLE_SESSION_REINITIATION	coReinitType
SESSION_LEVEL_CRYPTOGRAPHY	coSessCrypt
CONWINNER_AUTO_ACTIVATE_LIMIT	coDefPBFirstSpkrs
RETURN_CODE	appcHiResult
	appcLoResult

---



---

## DEFINE\_TP is CODEfineTP

LU 6.2 parameter	MacAPPC parameter
TP_NAME	coLclProgName
	coNetNameOp
	coNetName
STATUS	coEnableType
CONVERSATION_TYPE	coConvType
SYNC_LEVEL	coSyncType
SECURITY_REQUIRED	coSecReq
SECURITY_ACCESS	coSecOp
	coUserName
	coUserProf
PIP	coPIPReq
	coPIPCount
	coPIPCheck
DATA_MAPPING	coDataMapping
FMH_DATA	coFMHDataSupp
PRIVILEGE	coPrivType
RETURN_CODE	appcHiResult
	appcLoResult



---

---

## DISPLAY\_LOCAL\_LU is CODisplayLocalLU

LU 6.2 parameter	MacAPPC parameter
FULLY_QUALIFIED_LU_NAME	coLclLUName coNetName coNetQual
RETURN_CODE	appcHiResult appcLoResult
LU_SESSION_LIMIT	coDefMaxSess
LU_SESSION_COUNT	coActLUSess
SECURITY	coConvSec coUserName coUserPswd coUserProf
MAP_NAMES	N/A
REMOTE_LU_NAMES	N/A
TP_NAMES	N/A

---

---

## DISPLAY\_REMOTE\_LU is CODisplayRemoteLU

LU 6.2 parameter	MacAPPC parameter
FULLY_QUALIFIED_LU_NAME	coNetName coNetQual
RETURN_CODE	appcHiResult appcLoResult
LOCALLY_KNOWN_LU_NAME	coRmtLUName
UNINTERPRETED_LU_NAME	N/A
INITIATE_TYPE	coQueueINITs
PARALLEL_SESSION_SUPPORT	coParSess
CNOS_SUPPORT	coCNOSALSName
SECURITY_ACCEPTANCE_LOCAL_LU	coLclSecAcc
SECURITY_ACCEPTANCE_REMOTE_LU	coRmtSecAcc
MODE_NAMES	N/A



---

---

## DISPLAY\_MODE is CODisplayMode

LU 6.2 parameter	MacAPPC parameter
FULLY_QUALIFIED_LU_NAME	coRmtLUName
MODE_NAME	coModeName
RETURN_CODE	appcHiResult
	appcLoResult
SEND_PACING_WINDOW	coSendPacing
RECEIVE_PACING_WINDOW	coRcvPacing
SEND_MAX_RU_SIZE_UPPER_BOUND	coMaxRUHiBound
SEND_MAX_RU_SIZE_LOWER_BOUND	coMaxRULoBound
RECEIVE_MAX_RU_SIZE_UPPER_BOUND	coMaxRUHiBound
RECEIVE_MAX_RU_SIZE_LOWER_BOUND	coMaxRULoBound
SYNC_LEVEL_SUPPORT	coSyncType
SINGLE_SESSION_REINITIATION	coSessReinit
SESSION_LEVEL_CRYPTOGRAPHY	coSessCrypt
CONWINNER_AUTO_ACTIVATE_LIMIT	coDefPBFirstSpkrs
LU_MODE_SESSION_LIMIT	coCurMaxSess
MIN_CONWINNERS	coCurMinFirstSpkrs
MIN_CONLOSERS	coCurMinBdrs
TERMINATION_COUNT	coTermCount
DRAIN_LOCAL_LU	coDrainLclLU
DRAIN_REMOTE_LU	coDrainRmtLU
LU_MODE_SESSION_COUNT	coActSess
CONWINNERS_SESSION_COUNT	coActFirstSpkrs
CONLOSERS_SESSION_COUNT	coActBdrs
SESSION_IDS	N/A

---

---

## DISPLAY\_TP is CODisplayTP

LU 6.2 parameter	MacAPPC parameter
TP_NAME	coLclProgName
	coNetName
RETURN_CODE	appcHiResult
	appcLoResult
STATUS	coEnableType
CONVERSATION_TYPE	coConvType
SYNC_LEVEL	coSyncType



**LU 6.2 parameter****MacAPPC parameter**

SECURITY\_REQUIRED

coSecReq

SECURITY\_ACCESS

coUserName

coUserProf

PIP

coPIPReq

coPIPCount

coPIPCheck

DATA\_MAPPING

coDataMapping

FMH\_DATA

coFMHDataSupp

PRIVILEGE

coPrivType

---

---

**DELETE is CODElete****LU 6.2 parameter****MacAPPC parameter**

LOCAL\_LU\_NAME

LclLUName

REMOTE\_LU\_NAME

coRmtLUName

MODE\_NAME

coModeName

TP\_NAME

coLclProgName

RETURN\_CODE

appcHiResult

appcLoResult





## Appendix G



### MacAPPC Result Codes Mapping

This appendix gives a complete listing of the mapping between the parameters for each LU 6.2 return code and its corresponding MacAPPC result code. *N/A* indicates that there is no corresponding MacAPPC result code.



---

---

## Conversation return codes

### LU 6.2 return code

ALLOCATION\_ERROR  
ALLOCATION\_FAILURE\_NO\_RETRY  
ALLOCATION\_FAILURE\_RETRY  
CONVERSATION\_TYPE\_MISMATCH  
PIP\_NOT\_ALLOWED  
PIP\_NOT\_SPECIFIED\_CORRECTLY  
SECURITY\_NOT\_VALID  
SYNC\_LEVEL\_NOT\_SUPPORTED\_BY\_LU  
SYNC\_LEVEL\_NOT\_SUPPORTED\_BY\_PGM  
TPN\_NOT\_RECOGNIZED  
TRANS\_PGM\_NOT\_AVAIL\_NO\_RETRY  
TRANS\_PGM\_NOT\_AVAIL\_RETRY  
BACKED\_OUT  
DEAKKICATE\_ABEND  
DEALLOCATE\_ABEND\_PROG  
DEALLOCATE\_ABEND\_SVC  
DEALLOCATE\_ABEND\_TIMER  
DEALLOCATE\_NORMAL  
FMH\_DATA\_NOT\_SUPPORTED  
HEURISTIC\_MIXED  
MAP\_EXECUTION\_FAILURE  
MAP\_NOT\_FOUND  
MAPPING\_NOT\_SUPPORTED  
OK  
DATA  
NOT\_DATA  
PARAMETER\_ERROR  
POSTING\_NOT\_ACTIVE  
PROG\_ERROR\_NO\_TRUNC  
PROG\_ERROR\_PURGING  
PROG\_ERROR\_TRUNC  
SVC\_ERROR\_NO\_TRUNC  
SVC\_ERROR\_PURGING  
SVC\_ERROR\_TRUNC  
RESOURCE\_FAILURE\_NO\_RETRY  
RESOURCE\_FAILURE\_RETRY  
UNSUCCESSFUL

### MacAPPC result code

allocErr  
allocNoRetryErr  
allocRetryErr  
convTypeMatchErr  
pipSuppErr  
pipSpecErr  
secInfoErr  
N/A  
syncTypeSuppErr  
unkProgNameErr  
tpNoRetryErr  
tpRetryErr  
N/A  
abendErr  
abendProgErr  
abendSvcErr  
abendTimerErr  
normDeallocErr  
fmhSuppErr  
N/A  
mapProcErr  
mapNameErr  
mapSuppErr  
noErr  
dataAvail  
ctlAvail  
usageErr  
postActErr  
progNoTruncErr  
progPurgingErr  
progLLTruncErr  
svcNoTruncErr  
svcPurgingErr  
svcLLTruncErr  
rsrcErr  
rsrcErr  
badComplErr



---

---

## Control operator return codes

### LU 6.2 return code

### MacAPPC result code

ACTIVATION_FAILURE_NO_RETRY	sessActErr
ACTIVATION_FAILURE_RETRY	noSessSpaceErr
ALLOCATION_ERROR	allocErr
ALLOCATION_FAILURE_NO_RETRY	allocNoRetryErr
ALLOCATION_FAILURE_RETRY	allocRetryErr
TRANS_PGM_NOT_AVAIL_NO_RETRY	tpNoRetryErr
COMMAND_RACE_REJECTED	raceCNOSErr
LU_MODE_SESSION_LIMIT_CLOSED	limitsClosedErr
LU_MODE_SESSION_LIMIT_EXCEEDED	limitsTooBigErr
LU_MODE_SESSION_LIMIT_NOT_ZERO	limitsNotZeroErr
LU_MODE_SESSION_LIMIT_ZERO	limitsZeroErr
LU_SESSION_LIMIT_EXCEEDED	N/A
OK	noErr
AS_SPECIFIED	normalCompl
AS_NEGOTIATED	negotCompl
FORCED	N/A
PARAMETER_ERROR	usageErr
REQUEST_EXCEEDS_MAX_ALLOWED	limitsTooBigErr
RESOURCE_FAILURE_NO_RETRY	rsrcCNOSErr
UNRECOGNIZED_MODE_NAME	unkModeErr









## Appendix H



### HyperCard APPC

This appendix describes HyperCard APPC™, a HyperCard® stack that is included with MacAPPC. Some of the information in this appendix is also available in the stack itself.

HyperCard APPC provides a HyperCard-based MacAPPC

- ☐ development environment
- ☐ interactive lab with help
- ☐ sample application

As a development environment, HyperCard APPC supplies many of the basic handlers needed to create a HyperCard front end for MacAPPC. You can copy these handlers and use them in your own HyperCard MacAPPC applications. HyperCard APPC also supplies the external commands (XCMDs) and external functions (XFCNs) necessary to pass data between HyperCard and MacAPPC. As with the basic handlers, you are free to copy and use the XCMDs and XFCNs in your own applications.

The HyperCard APPC stack includes a “laboratory” for experimenting with and learning about MacAPPC routines. The MacAPPC Lab section is composed of a series of cards. Each card represents one MacAPPC routine and shows all of that routine's parameters. Pop-up help fields explain the purpose of the routine and describe each of the parameters. You can execute a specific MacAPPC routine handler after setting the values of the parameters that are passed to the MacAPPC driver (supplied parameters).

The sample application is an implementation of a HyperCard-based MacAPPC application called *APPC Mail*. APPC Mail is actually two applications in one: a server application called *Postmaster* and a client application called *Mailbox*. Postmaster allows you to select a picture and type in a message. Mailbox lets you request the current picture and message from the Postmaster. APPC Mail helps you understand MacAPPC conversation flows by displaying the conversation transactions as they occur. You can also study the APPC Mail scripts to see how the HyperCard APPC handlers, XCMDs, and XFCNs were used to implement the APPC Mail application.

This appendix assumes that you already are familiar with HyperCard, HyperTalk™ programming, and Stackware™. If you are not, you should read the *HyperCard User's Guide* and APDA's *HyperCard Script Language Guide* and experiment with some other HyperCard stacks.



---

---

## Setup

Chapter 9, "Installation," explains in detail the hardware and software requirements for running MacAPPC applications. This section reviews those requirements specifically related to using HyperCard APPC.

---

### Important

Be sure to use a working copy of HyperCard APPC. Never work with your original HyperCard APPC stack.

---

---

## Physical requirements

The sample session and sample application, described later in this appendix, assume conversations between two Macintosh computers. One of the computers must be a Macintosh II, with the intelligent communications card (server card) installed. Although the Macintosh II hosts the MacAPPC server, it is not dedicated to that process; it can simultaneously host the server *and* run its own applications. The other computer can be a Macintosh Plus, SE, or II. Because HyperCard is such a large application, both computers must have at least 2 megabytes of RAM. The computers should be connected via an AppleTalk network system.

---

## Software requirements

In order to use the HyperCard APPC stack, you must, of course, have the HyperCard application.

---

### Important

HyperCard APPC requires version 1.2 (or later) of the HyperCard application.

---

The MacAPPC product requires that several files be located in the System Folder of the Macintosh II computer with the server card, and that a subset of those files be in the System Folder of any other computer acting as a client. See "Software" in Chapter 9.

## Starting the MacAPPC server

The MacAPPC server must be started and the session limits activated for at least one user mode prior to running HyperCard APPC. See "Starting a MacAPPC Server" and "Displaying Network Components and Sessions" in Chapter 12.



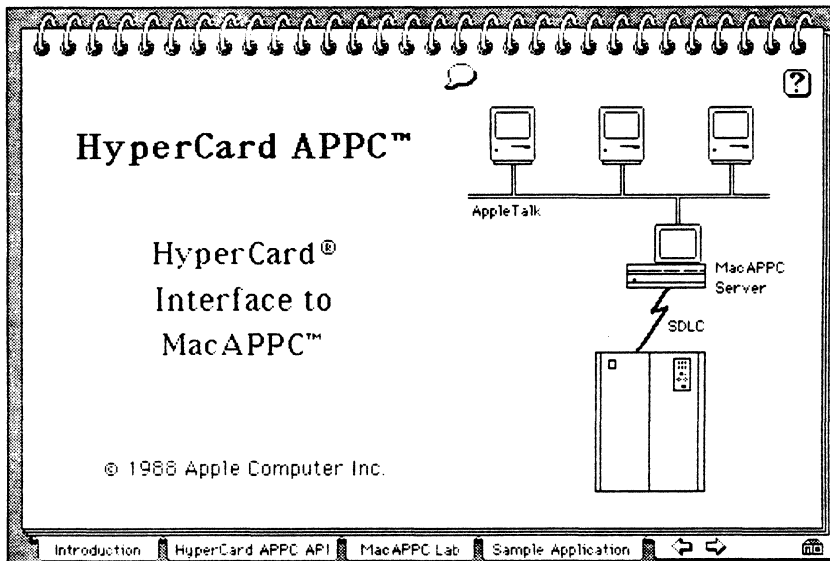
---

## Overview

The HyperCard APPC stack consists of the following sections:

- ☐ Introduction
- ☐ HyperCard APPC API
- ☐ MacAPPC Lab
- ☐ Sample Application

Figure H-1 shows the title card of the HyperCard APPC stack. The index tabs at the bottom of the card are buttons that take you to each section.



**Figure H-1**  
The HyperCard APPC stack title card

---

## Introduction

The introduction is a set of cards that describe the stack and its various components. These cards provide a brief introduction to APPC, MacAPPC, and HyperCard APPC programming. The introduction also tells you how to *navigate* (move around) the stack, explains how to get help, and gives overviews of the major sections of the stack.

---

## HyperCard APPC Application Programming Interface (API)

HyperCard APPC supplies the basic handlers you need to create HyperCard-based MacAPPC applications and the external commands (XCMDs) and external functions (XFCNs) that pass data between HyperCard and MacAPPC. The handlers, XCMDs, and XFCNs are known collectively as the *HyperCard APPC Application Programming Interface (API)*.



The HyperCard APPC API cards outline the process of creating your own HyperCard APPC applications using the HyperCard APPC API. "Developing HyperCard APPC Applications," later in this appendix, discusses the same topic.

---

## MacAPPC Lab

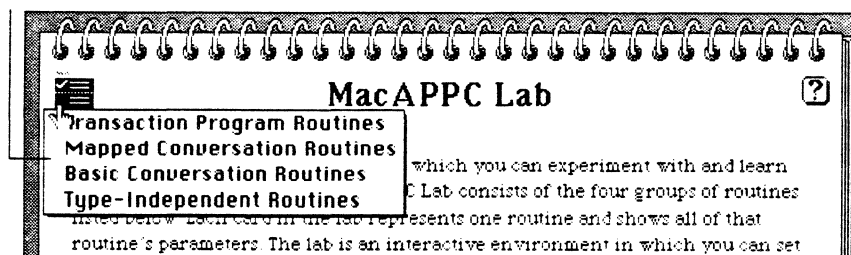
The MacAPPC Lab consists of four groups of cards. Each group contains a related set of MacAPPC routines, one routine per card. The four groups are

- ❑ transaction program routines (TP)
- ❑ mapped conversation routines (MC)
- ❑ basic conversation routines (BC)
- ❑ type-independent conversation routines (CV)

❖ *Note:* The HyperCard APPC Lab does not include control operator or node operator routine cards or handlers. Control operator and node operator are supported, however, by the APPC XCMD. (See "APPC XCMD," later in this appendix.)

The title card at the beginning of the MacAPPC Lab section lists the four groups of cards. Click one of the buttons to go directly to the named group. You can also go to one of the groups of routines by using the navigation button in the top-left corner of each card. When you press the navigation button, a pop-up menu appears from which you can choose one of the groups of routines. (See Figure H-2.)

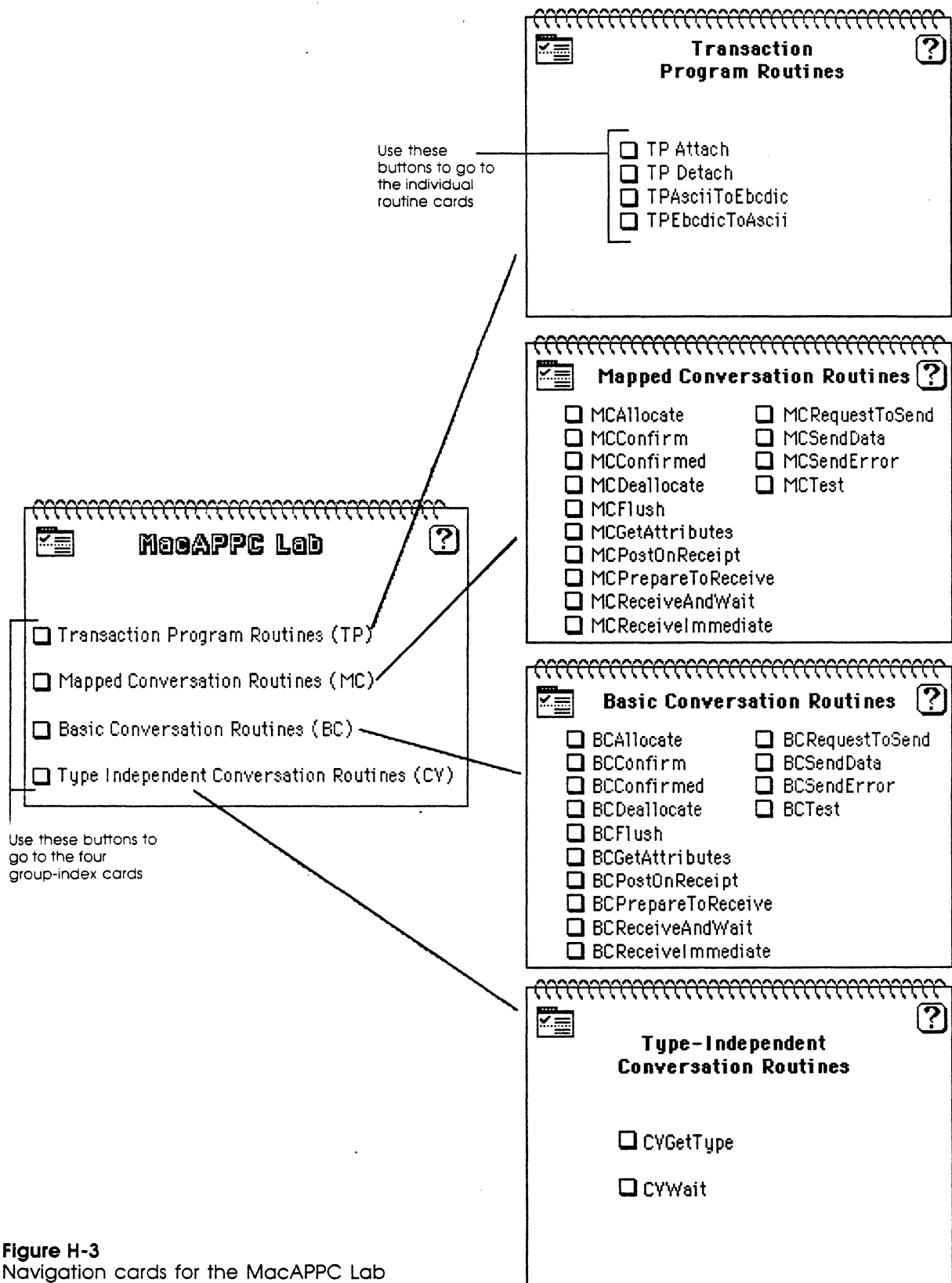
Use the navigation button's  
pop-up menu to go to  
another card in the lab



**Figure H-2**  
The navigation button pop-up menu

Similarly, each group of routines begins with an index of the individual routine cards contained in that group. Click the button next to a routine name to go directly to its card. Figure H-3 shows the MacAPPC Lab title card and the four subordinate group-index cards.





**Figure H-3**  
Navigation cards for the MacAPPC Lab



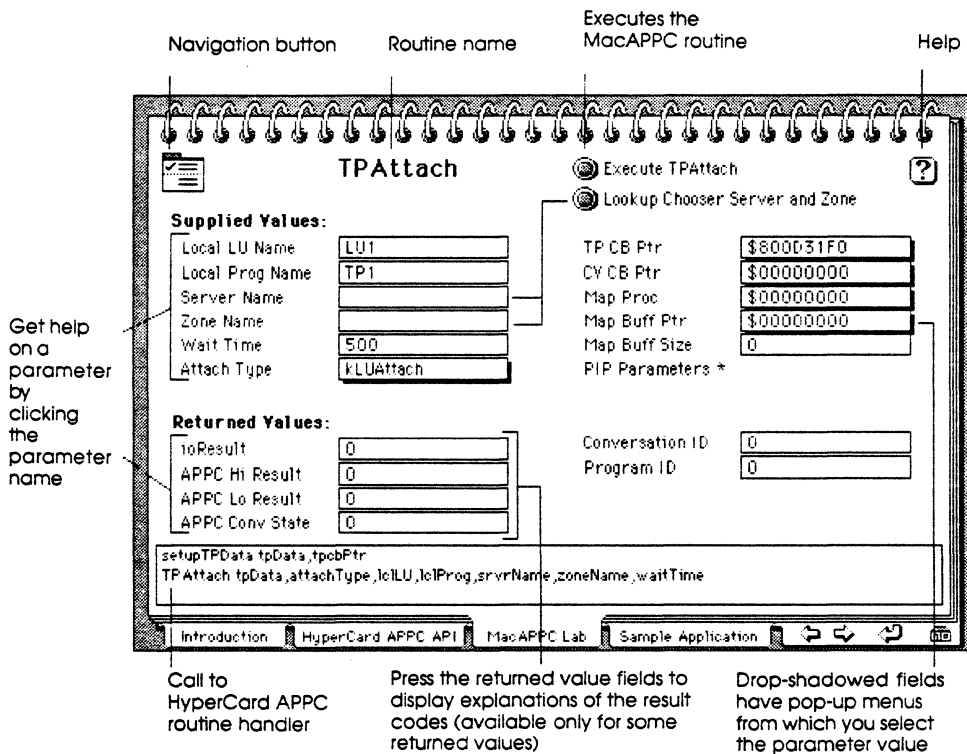
- ❖ *Routines versus verbs*: MacAPPC conversation routines correspond directly to APPC verbs. MacAPPC transaction program routines, however, are unique to the MacAPPC product, and have no APPC counterparts.

## MacAPPC routine cards

The MacAPPC routine cards perform several functions. Each card

- displays the routine's parameters and lets you enter or change the supplied parameters
- teaches about the parameters through the built-in help system
- displays a script showing the parameter list
- lets you execute the routine
- provides HyperTalk scripts that you can copy for your own HyperCard applications

Figure H-4 illustrates the elements of MacAPPC Lab cards using the TPAttach card as an example. The name of the routine appears at the top, a little left of center. The navigation button is in the top-left corner. The help button appears in the top-right corner. Most of the card is taken up by supplied and returned value fields. The text field across the bottom of the card displays the call to the HyperCard APPC routine handler. You execute the routine by clicking the Execute button at the top of the card, to the right of the routine name. (The Lookup Chooser Server and Zone button is unique to the TPAttach routine card. See "Other Elements," later in this appendix.)



**Figure H-4**  
TPAttach routine card



## Supplied Values parameters

*Supplied Values* are the parameters passed to the MacAPPC drivers when a MacAPPC routine is executed. The routine cards in the MacAPPC Lab let you set those parameters, either by entering them in the appropriate text fields or by choosing items from pop-up menus. Pop-up menus are indicated by the drop-shadowed parameter fields, as shown in Figure H-4. Press the parameter field to display the menu; drag to make a selection.

You can get information about a particular parameter by clicking the parameter name. See “Lab Help,” later in this appendix.

HyperCard APPC provides default values for some of the supplied parameters. You can accept the default values or enter your own. Other values HyperCard APPC allocates for you, and cannot be set directly. HyperCard APPC automatically allocates values for

- ☐ control block pointers (TP CB Ptr, CV CB Ptr)
- ☐ buffer pointers (Map Buff Ptr, Data Ptr)
- ☐ buffer sizes (Map Buff Size, Data Size)
- ☐ mapping procedure pointer (Map Proc)

HyperCard APPC supplies two TP CB Ptrs and two CV CB Ptrs that support two attaches with one conversation each, or one attach with two conversations. Pop-up menus let you choose between the two conversations.

After you set the supplied values, you can click the Execute button at the top of the card to execute the routine.

**TPAttach card:** When you choose kLUAttach (the default) for the Attach Type parameter on the TPAttach card, the following parameters are ignored:

- ☐ CV CB Ptr
- ☐ Map Proc
- ☐ Map Buff Ptr

The parameters become active when you choose kWaitAttach for the Attach Type parameter. Pop-up menus let you choose between two conversations (CV CB Ptr parameter); no mapping procedure or a sample mapping procedure (Map Proc parameter); and the supplied map buffer for a 256-byte buffer, plus 4 bytes of overhead, or no buffer (Map Buffer parameter). Because these parameters are pointers, HyperCard APPC allocates the actual values.

## Returned Values parameters

*Returned Values* are the parameters yielded when a MacAPPC routine call is completed. The returned values provide result codes, the current conversation state, and other information.

You can get information about a returned value parameter by clicking the parameter name. You can also get help about specific result codes by pressing the returned value (moving the pointer over the value and holding the mouse button down). See “Lab Help,” later in this appendix.



The ioResult field tells you whether or not an error occurred. If 0 (zero) appears, there was no error; if -5000 appears, an error did occur. When an error occurs, the APPC Hi Result field tells you the type of error and APPC Lo Result gives you details. Table H-1 lists the major error result codes (APPC Hi Result). See also Appendix C, "MacAPPC Result Codes," which lists the major error result codes and the minor error result codes (APPC Lo Result).

**Table H-1**  
APPC major errors (APPC Hi Result)

APPC Hi Result value	Description
0	Function completed normally
1	Function aborted, usage error
2	Function not completed
3	Function aborted, state error
5	Function aborted, allocation error
7	Program error
9	Deallocated
10	Control operator error
11	Node operator error

The value returned in the APPC Conv State field gives the current conversation state of your node. Table H-2 lists the state values and their meanings. (See also "appcConvState" in Chapter 3 and "Conversation States" in Chapter 4.)

**Table H-2**  
APPC conversation state

APPC Conv State value	Description
0	Null
1	Reset
2	Send
3	Receive
4	Confirm
5	Confirm send
6	Confirm deallocate
7	Deallocate



## Other elements

This section describes the elements found only on one or a few of the routine cards.

**Conversation ID and Program ID fields:** Program ID and Conversation ID appear on the TPAttach, BCGetAttributes, and MCGetAttributes cards. Conversation ID also appears on the MCAAllocate and BCAllocate cards.

On the TPAttach card, the Conversation ID is valid only when the attach type is kWaitAttach.

**Send data field with pop-up menu and Save button:** The TPAsciiToEbcDic, TPEbcDicToAscii, MCSendData, BCDeallocate, BCSendData, and BCSendError cards contain a scrolling field in which you can enter data to translate or send. A pop-up menu lets you display the data as text or as hexadecimal code. The Save button lets you save the data at the location pointed to by the data pointer (TP Data Ptr or Data Ptr).

---

### Warning

Because \$00 (hexadecimal code 00) is the terminator for HyperCard strings, do not display hexadecimal data as text if it contains \$00. All data after \$00 is lost when displayed as text.

---

**Receive data field with pop-up menu:** The MCRceiveAndWait, BCReceiveAndWait, MCRceiveImmediate, and BCReceiveImmediate cards contain a scrolling field that displays the data currently pointed to by the data pointer (Data Ptr). (This is the same pointer and buffer used by the TPAsciiToEbcDic, TPEbcDicToAscii, MCSendData, and BCSendData cards.) A pop-up menu lets you display the data as text or as hexadecimal code.

❖ *Note:* Because \$00 (hexadecimal code 00) is the terminator for HyperCard strings, text display will not show data after \$00.

**Lookup Chooser Server and Zone button:** Clicking the Lookup Chooser Server and Zone button on the TPAttach card enters in the appropriate Supplied Values fields (Server Name and Zone Name) with the MacAPPC server and zone names currently specified by the Chooser. Or you can enter any other server and zone names of your choice.

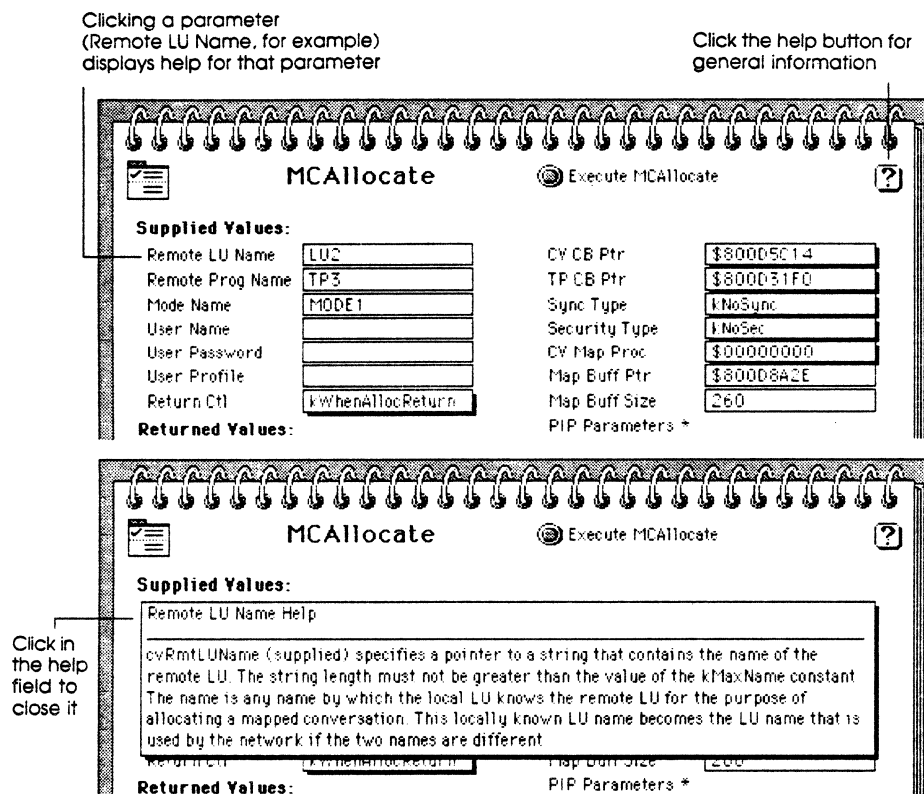
### Lab Help

Each card has a help button in the top-right corner. Clicking the button displays a general description of the routine.



Each card also contains help fields for each of the routine's individual parameters (both supplied and returned). To display a parameter's help field, click the parameter name. For example, on the MCAAllocate card, clicking Remote LU Name under Supplied Values displays the Remote LU Name help field. (See Figure H-5.) Click anywhere in the help field to close it.

❖ *Note:* The help fields on the routine cards contain the same text as is used in Part II of this guide.



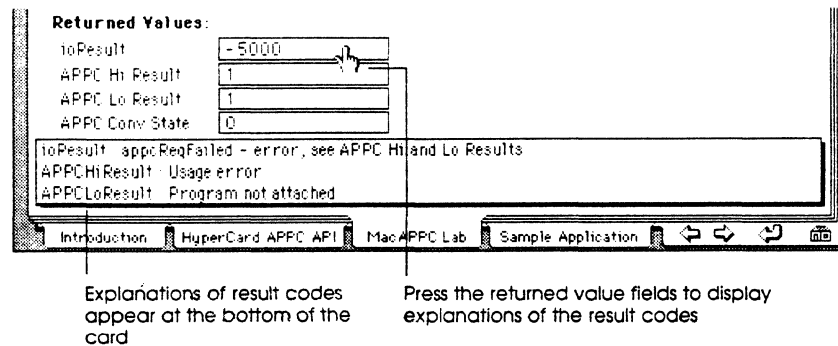
**Figure H-5**  
Remote LU Name help field

In addition, descriptions of the actual result codes returned are available for the following parameters:

- ☐ ioResult
- ☐ APPC Hi Result
- ☐ APPC Lo Result
- ☐ APPC Conv State
- ☐ What Rcvd
- ☐ Req To Send Rcvd
- ☐ Sync Type
- ☐ Conversation Type



To see the explanation of a result code, position the pointer over the value and hold the mouse button down. The explanation appears at the bottom of the window, as shown in Figure H-6.



**Figure H-6**  
Help on error result codes

## Sample Application

The sample application—APPC Mail—demonstrates a simple conversation between two Macintosh computers using HyperCard APPC. This sample application is really two applications in one: a server application (Postmaster) and a client application (Mailbox). Postmaster sends messages and pictures to Mailbox. To conduct a conversation using APPC Mail, one node must be running Postmaster and the other must be running Mailbox.

APPC Mail lets you learn about MacAPPC conversation flows by displaying a log of the MacAPPC routines as they are executed. Both server and client routines are listed, as explained in the sections that follow.

You can also learn about creating HyperCard APPC applications by studying the background scripts and card scripts used in the sample application. The scripts use the HyperCard APPC API. (See “HyperCard APPC API,” earlier in this appendix.) These scripts give examples that you can use in your own application.

Before you can use APPC Mail, you must do the following:

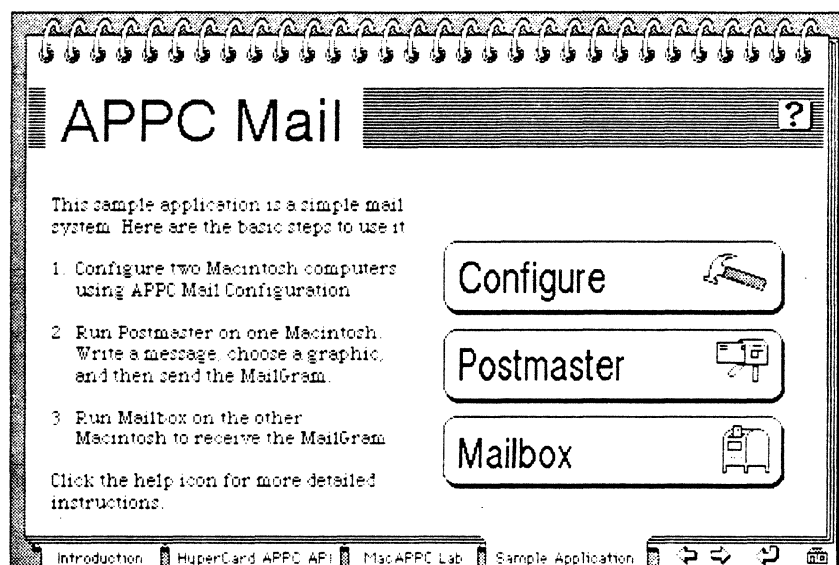
1. Start the server using Admin and the Local Config configuration document. The two files are on the *MacAPPC User* disk. (See “Starting a MacAPPC Server” in Chapter 12.)
2. Activate the LUs. (See “Activating Network Components and Sessions” in Chapter 12.)
3. Start a CNOS. (See “Starting and Stopping CNOS” in Chapter 12.)
4. Activate the mode limits. (See “Activating Network Components and Sessions” in Chapter 12.)
5. Select the MacAPPC server (with the Chooser) for both clients. (See “Selecting a MacAPPC Server” in Chapter 10.)

Figure H-7 shows the APPC Mail title card. The three large buttons on the card let you

- ☐ configure the local node
- ☐ start up Postmaster (server application)
- ☐ start up Mailbox (client application)



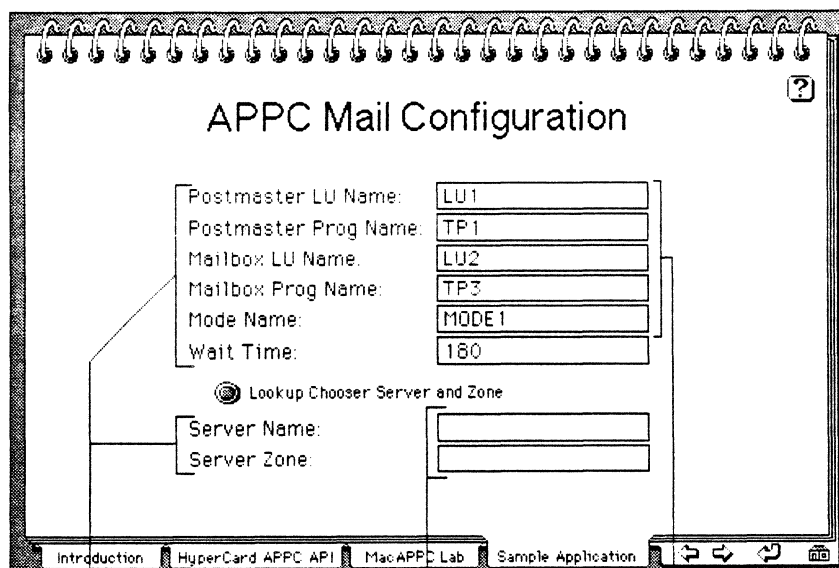
You perform these actions by clicking the appropriate button.



**Figure H-7**  
APPC Mail title card

### APPC Mail Configuration

Clicking the Configure button takes you to the configuration card. APPC Mail comes preconfigured, as shown in Figure H-8. APPC Mail uses the values on this card for the names of the logical units, transaction programs, and mode. It also provides the value for the time-out for a response to the `MCReceiveAndWait` routine (Wait Time parameter).



Get help on a parameter by clicking the parameter name"

If these fields are empty and you have selected a MacAPPC server from the Chooser, APPC Mail automatically fills in the server name and zone when the stack is opened

These values work with the "Local Config" file

**Figure H-8**  
APPC Mail configuration card

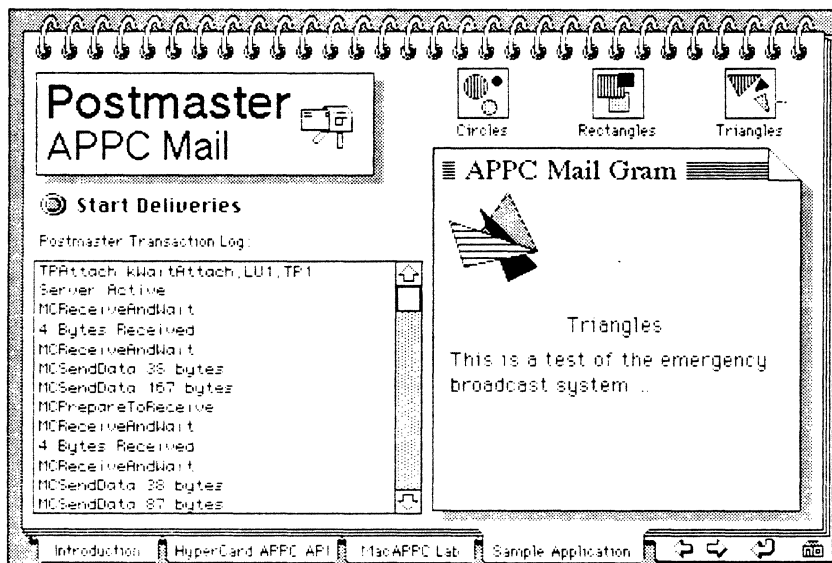


HyperCard APPC automatically looks up the server and zone names when the stack is opened, provided that you have previously chosen a MacAPPC server and neither field already had a value. Otherwise, you need to select a server name and zone from the Chooser, and then click the Lookup Chooser Server and Zone button. Or, type in the name and zone of the server you wish to use. (Note that case is significant.) You will also need to click the Lookup Chooser Server and Zone button if, in the future, you choose a different MacAPPC server.

Thereafter, it is usually not necessary to configure this application—the default values should work under most conditions. Do not change these values unless you are sure you need to. Any changes you make will remain until you change the values again. If you do make changes, first copy the values shown in Figure H-8, or make a new working copy of the HyperCard APPC stack from your original in case you want to return to the original settings.

## Postmaster

You start Postmaster by clicking the Postmaster button on the APPC Mail title card. The Postmaster card is shown in Figure H-9.



**Figure H-9**  
APPC Mail Postmaster

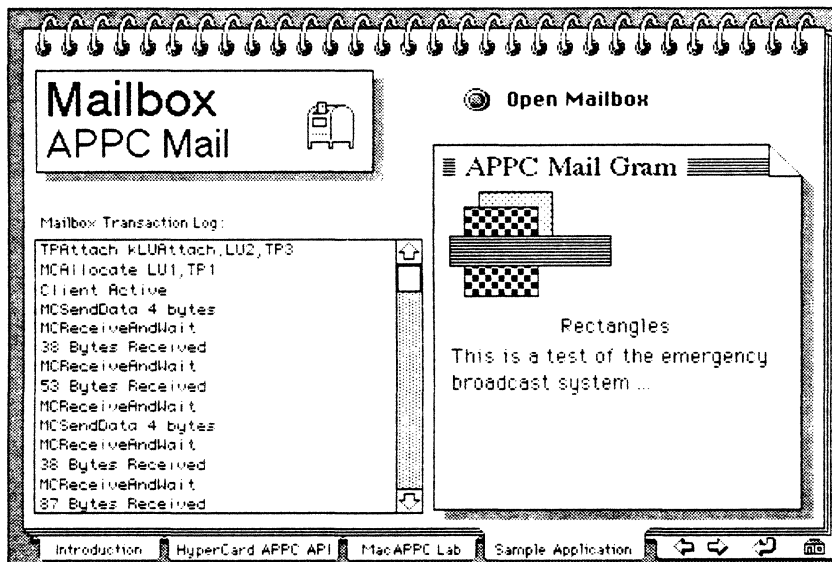
Postmaster lets you create APPC Mail Grams and send them to the Mailbox application. An APPC Mail Gram consists of a block of text and a picture. You can send one of three pictures: circles, rectangles, or triangles. Click one of the buttons above the Mail Gram to make your choice. The picture you are sending appears at the top of the Mail Gram, and the picture's title appears in the center. You can edit the text block and create your own message in the usual Macintosh way.



When you have chosen a picture and entered your message, you are ready to send the Mail Gram. Click the Start Deliveries button. Postmaster will wait for a request for data from the Mailbox. When the request is received, Postmaster sends the picture and text to Mailbox. As MacAPPC routines are called, Postmaster logs the routines and their responses in the scrolling text field on the left side of the card.

## Mailbox

Make sure Postmaster is started first. Then click the Mailbox button on the APPC Mail title card to start the Mailbox client application. Figure H-10 shows the Mailbox card. Clicking the Open Mailbox button at the top of the card sends a request for data to Postmaster. Mailbox will then receive the Mail Gram you created with Postmaster. (See the previous section.)



**Figure H-10**  
APPC Mail Mailbox

As with the Postmaster application, a log of the MacAPPC routines and responses appears in the scrolling text field on the left side of the card.



---

---

## Sample session: Stepping through a conversation

The following exercise shows how you can use the MacAPPC Lab to conduct a conversation between two MacAPPC programs. The exercise establishes a conversation between two clients of a MacAPPC server. Both clients must first attach to their local LU. One of the clients then allocates a conversation and sends a message to the other client. That client receives the message and sends a reply. After the first client receives the second client's reply, both clients deallocate the conversation and detach from the local LU.

Two 2-megabyte Macintosh computers connected via AppleTalk are required. (See "Setup" earlier in this appendix.) Before an actual conversation can take place you must do the following:

1. Start the server using Admin and the Local Config configuration document. The two files are on the *MacAPPC User* disk. (See "Starting a MacAPPC Server" in Chapter 12.)
2. Activate the LUs. (See "Activating Network Components and Sessions" in Chapter 12.)
3. Start a CNOS. (See "Starting and Stopping CNOS" in Chapter 12.)
4. Activate the mode limits. (See "Activating Network Components and Sessions" in Chapter 12.)
5. Select the MacAPPC server (with the Chooser) for both clients. (See "Selecting a MacAPPC Server" in Chapter 10.)

After making sure that the above conditions are met, you can use the MacAPPC Lab routine cards to conduct the conversation. Designate one of the Macintosh computers Client A (TP1), the other Client B (TP3). Then follow the instructions below.



## Client A

### ■ Attach to the local LU and wait to be allocated

1. Go to the TPAttach card.
2. Set the parameters:
  - a. Type "LU1" in the Local LU Name box.
  - b. Type "TP1" in the Local Prog Name box.
  - c. Click the Lookup Chooser Server and Zone button to fill in the Server Name and Zone Name fields.
  - d. Type "180" in the Wait Time box.
  - e. Choose kWaitAttach from the Attach Type pop-up menu.

*The MacAPPC Lab automatically allocates the TP CB Ptr, the CV CB Ptr, and the Map Buff Ptr; sets the Map Buff Size; and sets the Map Proc to the default of \$00000000.*

3. Click the Execute TPAttach button.

*The TPAttach will not return until Client B executes MCPPrepareToReceive, at which time the returned parameters will be filled in. (See "Prepare to Receive Data" under Client B.)*

## Client B

### ■ Attach to the local LU

1. Go to the TPAttach card.
2. Set the parameters:
  - a. Type "LU2" in the Local LU Name box.
  - b. Type "TP3" in the Local Prog Name box.
  - c. Click the Lookup Chooser Server and Zone button to fill in the Server Name and Zone Name fields.
  - d. Type "180" in the Wait Time box.
  - e. Choose kLUAttach from the Attach Type pop-up menu.

*The MacAPPC Lab automatically allocates the TP CB Ptr, the CV CB Ptr, and the Map Buff Ptr; sets the Map Buff Size; and sets the Map Proc to the default of \$00000000.*

3. Click the Execute TPAttach button.



## Client B

---

### ■ Allocate a conversation with the partner application

1. Go to the MCAAllocate card.
2. Set the parameters:
  - a. Type "LU1" in the Remote LU Name box.
  - b. Type "TP1" in the Remote Prog Name box.
  - c. Type "MODE1" in the Mode Name box.
2. Click the Execute MCAAllocate button.

### ■ Send data to Client A

1. Go to the MCSendData card.

*There's no need to set any parameters—you can use the default values.*
2. Type the message you want to send into the scrolling text field.

*(For example: "Hello TP1, how are you?")*
3. Click the Save Data button.

*You must save the text to a buffer before it can be sent.*
4. Click the Execute MCSendData button.

*The text won't actually be sent until some other routine flushes the send buffer and "turns the conversation around." The MCPPrepareToReceive routine, executed next, accomplishes these tasks.*

### ■ Prepare to receive data

*Flush the send buffer and put Client B into receive state.*

1. Go to the MCPPrepareToReceive card.

*There's no need to set any parameters.*
2. Click the Execute MCPPrepareToReceive button.



## Client A

---

### ■ Receive the message from Client B

1. Go to the MCRceiveAndWait card.

*There's no need to set any parameters—you can use the default values.*

2. Click the Execute MCRceiveAndWait button.

*The data sent by Client B appears in the text box, and the APPC Conv State value of 3 indicates that Client A is still in receive state.*

### ■ Make sure that there are no more messages

1. Go to the MCRceiveAndWait card.

*There's no need to set any parameters.*

2. Click the Execute MCRceiveAndWait button.

*The value of 5 in What Rcvd means that no data was received, and the transaction program needs to go into send state. The APPC Conv State value of 2 shows that Client A is in send state.*

### ■ Send a return message to Client B

1. Go to the MCSendData card.

*There's no need to set any parameters.*

2. Type a message into the scrolling text field.

*(For example: "Fine, thanks, TP3. How are you?")*

3. Click the Save Data button.

*You must save the text to a buffer before it can be sent.*

4. Click the Execute MCSendData button.

### ■ Flush the buffer and wait for response

1. Go to the MCRceiveAndWait card.

*There's no need to set any parameters.*

2. Click the Execute MCRceiveAndWait button.



## Client A

---

## Client B

---

### ■ Receive Client A's response

1. Go to the MCRceiveAndWait card.  
*There's no need to set any parameters.*
2. Click the Execute MCRceiveAndWait button.  
*Client A's message appears in the scrolling text field. The APPC Conv State value of 3 shows that Client B is in receive state.*

### ■ See if there are any more messages from TP1

1. Go to the MCRceiveAndWait card.  
*There's no need to set any parameters.*
2. Click the Execute MCRceiveAndWait button.  
*The APPC Conv State returned value of 2 shows that Client B is now in send state.*

### ■ Deallocate the conversation

1. Go to the MCDeallocate card.  
*There's no need to set any parameters.*
2. Click the Execute MCDeallocate button.

*The -5000 code in ioResult is normal when a conversation is deallocated. The APPC Hi Result should be 9, and the APPC Lo Result should be 0 (zero).*

### ■ Deallocate the conversation

1. Go to the MCDeallocate card.
2. Choose kLocalDealloc from the Dealloc Type pop-up menu.
3. Click the Execute MCDeallocate button.

### ■ Detach the client

1. Go to the TPDetach card.  
*There's no need to set any parameters.*
2. Click the Execute TPDetach button.

### ■ Detach the client

1. Go to the TPDetach card.  
*There's no need to set any parameters.*
2. Click the Execute TPDetach button.



---

---

## Developing HyperCard APPC applications

This section describes how to develop your own HyperCard APPC application. It contains information on the handlers, XCMDs, and XFCNs in the HyperCard APPC stack that you may use in your own HyperCard APPC application, and gives general guidelines on how to put it all together.

To start your application, use a copy of the HC APPC Starter Stack, which is included on the HyperCard APPC disk. The HC APPC Starter Stack contains all the necessary resources (XCMDs, XFCNs, and so forth) that you need for your own application. The stack consists of a single card that contains additional help on starting your application. You can delete this card once you start adding your own cards.

Before you start your application, you should consider the elements of all HyperCard APPC applications:

### ■ A user interface

Your application should have a standard Macintosh and HyperCard user interface. See the *HyperCard User's Guide*, *HyperCard Stack Design Guidelines*, *HyperCard Script Language Guide*, and *Human Interface Guidelines*.

### ■ Application-specific data structures and logic (algorithms)

You must develop the application logic by using the HyperTalk scripting language. Much of the work has already been done for you. Simply copy the relevant HyperCard APPC handlers. See "HyperCard APPC Scripts and Handlers," later in this appendix.

Your application also must be synchronized with its partner applications. You must define all data structures sent or received over APPC with the HyperCard APPC XData XCMDs. See "XData XCMDs and XFCNs" later in this appendix.

### ■ The MacAPPC routines

A single HyperCard XCMD, APPC, provides access to all of the MacAPPC routines, which in turn provide access to the APPC protocol. Handlers for each of the mapped conversation (MC), basic conversation (BC), type-independent conversation (CV), and transaction program (TP) routines fill in the required fields of the corresponding parameter block before calling the XCMD. You can use the HyperCard APPC routine handlers in your own scripts or have your own handlers call APPC directly.

The following sections list the XCMDs, XFCNs, and HyperCard handlers that you may find useful in your application.

---

## APPC XCMD

All of HyperCard APPC's routine handlers access the APPC protocol by calling APPC, a HyperCard XCMD. (See "Scripts of the Routine Cards," later in this chapter.) A routine handler passes to the APPC XCMD the routine name, the name of the record containing the routine parameters, and a Boolean argument allowing or preventing asynchronous execution.



The syntax of `APPC` is

`APPC routineName, routineData, [async]`

where

- `routineName` is the name of the MacAPPC routine to be executed
- `routineData` is the name of the data record (created by the XData XCMDs) that is used for passing parameters to the MacAPPC routine
- `async` (optional) is either TRUE or FALSE—a value of TRUE permits asynchronous execution of the routine; a value of FALSE, or no value, prevents asynchronous execution

Chapter 4, “MacAPPC Conversation Driver,” describes the mapped, basic, and type-independent conversation routines. Chapter 5, “MacAPPC Control Operator Driver,” describes the control operator routines. Chapter 6, “MacAPPC Node Operator Driver,” describes the node operator routines. Chapter 7, “MacAPPC Transaction Program Driver,” describes the transaction program routines.

---

## get62Srvr XCMD

The `get62Srvr` XCMD gets the current MacAPPC server name and zone selected by the Chooser.

The syntax of `get62Srvr` is

`get62Srvr entityPtr`

where

- `entityPtr` is a pointer to a variable of type `EntityName` (see *Inside Macintosh*, Volume II)

Note that it is not necessary to use the server selected by the Chooser. If you want, you may “hard-wire” the server name and zone into your code. Remember, however, that the server must be running.

The following is an example of how to use the `get62Srvr` XCMD. See “XData XCMDs and XFCNs,” later in this appendix, for descriptions of `xDefine`, `xGlobal`, `xLock`, and `xPtr`.

```
xDefine EntityNameRec
xDefine  ObjName,pstring,33
xDefine  TypeName,pstring,33
xDefine  ZoneName,pstring,33
xDefine endDef

xGlobal EntityName,EntityNameRec
xLock EntityName, on -- Need to lock the record before calling xPtr
get62Srvr xPtr(EntityName)
xLock EntityName, off
```

- ❖ *Note:* The `defineCVandTP` handler, in the script of the first card of the stack, defines `EntityNameRec`. You do not need to redefine `EntityNameRec` if you use the `defineCVandTP` handler in your application.



If you want your application to have a configuration card that allows users to select the server and zone, you can use the `GetChosenSrvr` handler. `GetChosenSrvr` uses the `get62Srvr` XCMD to lookup the Chooser server name and zone, then displays those values in fields on the configuration card. `GetChosenSrvr` is located in the card scripts of both the TPAttach card of the MacAPPC Lab, and the APPC Mail Configuration card of the Sample Application.

---

## xConst XFCN

The `xConst` XFCN looks up the values of constants used in MacAPPC. See Appendix A, "MacAPPC Interface File," for a list of these constants.

The syntax of `xConst` is

```
xConst ("constantName")
```

where

*constantName* is the name of the constant whose value you want to look up

Because `xConst` is a function, you should place the value returned into a HyperCard container, as in the following example:

```
put xConst("kAPPCSize") into paramBlockSize
```

---

## errStr XCMD

`errStr` looks up the major and minor result code error message strings as defined in Appendix C ("MacAPPC Result Codes") and returns them in HyperCard global variables. `errStr` is useful for displaying textual error messages instead of just result codes. This is the XCMD that gets the text of the result code explanations that appear when you press the `ioResult`, `APPC Hi Result`, or `APPC Lo Result` returned value fields on the routine cards in the MacAPPC Lab.

The syntax of `errStr` is

```
errStr(majorCode, minorCode, "majorGlobal", "minorGlobal")
```

where

*majorCode* is the major return code from the `appcHiResult` field of the APPC parameter block

*minorCode* is the minor return code from the `appcLoResult` field of the APPC parameter block

*majorGlobal* is the name of the HyperCard global variable that will be set to the error message string corresponding to *majorCode*

*minorGlobal* is the name of the HyperCard global variable that will be set to the error message string corresponding to *minorCode*

The *majorGlobal* and *minorGlobal* arguments must be enclosed in quotation marks or HyperCard passes `errStr` their values instead of their names.



---

## HyperCard APPC scripts and handlers

This section lists the scripts and handlers in the HyperCard APPC stack that you are most likely to need in developing your own applications. You should study them before you begin programming. Take special note of how the sample application (APPC Mail) was put together. Although limited in its functionality, APPC Mail illustrates the basic elements of a HyperCard-based MacAPPC front end, and there are several useful handlers in that part of the stack.

You are free to use any of the handlers and scripts in the HyperCard APPC stack that you find useful. You can find these handlers in the following places:

- ☐ the stack script
- ☐ the script of the first card of the stack
- ☐ the script of the background for the routine cards (MacAPPC Lab)
- ☐ the scripts of each of the routine cards (MacAPPC Lab)
- ☐ the background script of the sample application (APPC Mail)
- ☐ the Postmaster card script (APPC Mail)
- ☐ the Mailbox card script (APPC Mail)

In this section, only HyperTalk language elements such as handler names, XCMDs, and XFCNs appear in computer voice. MacAPPC terms appear in the normal type face even though they appear in computer voice in all other parts of this guide.

### Stack script

The important handler in the stack script is `openStack`, which calls the `defineCVandTP` handler located in the script of the first card of the stack. See the next section, "Script of the First Card."

### Script of the first card

The first card of the HyperCard APPC stack contains two important handlers: `defineCVandTP` and `defineCOandNO`. These handlers define data structures, as described in the following subsections. The handlers also define record types for the MacAPPC driver parameter block, `APPCParamBlock`.

**defineCVandTP:** `defineCVandTP` defines the important data structures used by HyperCard APPC. Specifically, `defineCVandTP` defines the following data types:

<code>APPCCVRec</code>	used for passing parameters to conversation routines*
<code>APPCTPRec</code>	used for passing parameters to transaction program routines*
<code>tpStrRec</code>	record of strings that will be pointed to by fields in <code>APPCTPRec</code> *
<code>cvStrRec</code>	record of strings that will be pointed to by fields in <code>APPCCVRec</code> *
<code>CVCBRec</code>	defines the data structure used for conversation control blocks (CVCBs)*
<code>TPCBRec</code>	defines the data structure used for transaction program control blocks (TPCBs)*



EntityNameRec same as the AppleTalk EntityName record—see “get62Srvr XCMD,” earlier in this appendix

bcRec used to hold data for sending and receiving routines

dataRec assorted usages including declaration of MapProc and MapBuff

• Used by almost all HyperCard APPC applications.

defineCVandTP also allocates and locks space for TPCBs, CVCBs, MapBuff, and MapProc, and loads in MapProc. You will probably want to copy the defineCVandTP handler into your own HyperCard APPC application and then modify it to suit your needs.

**defineCOandNO:** defineCOandNO defines the data structures for the control operator (CO) and node operator (NO) routines. Specifically, defineCOandNO defines the following data types:

APPCCORec used for passing parameters to control operator (CO) routines

APP\_CNORec used for passing parameters to node operator (NO) routines

The HyperCard APPC stack does not actually use the defineCOandNO handler, but it exists in case your application uses CO or NO routines.

**APPCCParamBlock:** The data structure APPCCParamBlock passes parameters to and from the MacAPPC drivers. (See “MacAPPC Driver Parameter Block” in Chapter 3.) A variant part of the record allows each of the four MacAPPC drivers to use the same data structure. Because xDefine does not allow you to define variant records, however, you must define a different XData record type for each case of the APPCCParamBlock. (See “XData XCMDs and XFCNs,” later in this appendix.) The script of the first card of the stack defines these record types. The defineCVandTP handler defines APPCCVRec and APPCTPRec. The defineCOandNO handler defines APPCCORec and APP\_CNORec. Table H-3 lists the HyperCard APPC record types and the corresponding Pascal cases of the APPCCParamType tag field of the APPCCParamBlock record. The HyperCard APPC record-type field names have the same names and data types as in the Pascal record.

**Table H-3**  
HyperCard APPC record types

Type name	Pascal case of APPCCParamBlock RECORD
APPCCVRec	CVParam
APPCTPRec	TPParam
APPCCORec	COParam
APP_CNORec	NOParam



## Routine cards' background script

The background for the MacAPPC Lab's routine cards contains four important handlers:

- `attachState`      called by transaction program routines to allocate an APPCTPRec record type named `tpData`, if it is not already allocated
- `allocateState`   called by conversation routines to allocate an APPCCVRec record type named `convData`, if it is not already allocated
- `setupCVData`      called by conversation routines to set the following fields in `convData`:
  - ☐ `cvTPCBPtr`
  - ☐ `cvCVCBPtr`
  - ☐ `cvMapProc`
  - ☐ `cvMapBuffPtr`
  - ☐ `cvMapBuffSize`
- `setupTPData`      called by transaction program routines to set the following fields in `tpData`:
  - ☐ `tpTPCBPtr`
  - ☐ `tpCVCBPtr`
  - ☐ `tpMapProc`
  - ☐ `tpMapBuffPtr`
  - ☐ `tpMapBuffSize`

## Scripts of the routines cards

The scripts of the MacAPPC Lab routine cards contain two types of handlers:

- ☐ routine handlers
- ☐ exec handlers

For each routine card there is a routine handler that calls the APPC XCMD. The APPC XCMD is a single HyperCard XCMD that provides access to all of the MacAPPC routines. (See "APPC XCMD," earlier in this appendix.) The routine handler has the same name as the card's routine. For example, the MCAAllocate routine card script contains a handler named `MCAAllocate`, which passes the MCAAllocate routine parameters to the APPC XCMD. The APPC XCMD, in turn, calls the corresponding MacAPPC routine. Most HyperCard APPC applications will use the routine handlers.

The name of the exec handler for a routine is `execroutineName`, where *routineName* is the name of the card's routine. For example, the MCAAllocate exec handler is called `execMCAAllocate`. The exec handler is called when the Execute button on the routine card is pressed. An exec handler calls the `setupCVData` handler (or the `setupTPData` handler for TP routines), and then calls the appropriate routine handler. ("Routine Cards' Background Script," earlier in this appendix, describes `setupCVData` and `setupTPData`.)



## Background script of the sample application

The APPC Mail background script contains these important handlers:

defineType	defines the following data types: <ul style="list-style-type: none"><li>□ responseRec (the record for data sent to Mailbox)</li><li>□ requestRec (the record for data sent to Postmaster)</li><li>□ dataRec (used for sending PICT and TEXT data)</li></ul> and allocates space for the following: <ul style="list-style-type: none"><li>□ sampleTPCB (the TPCB)</li><li>□ sampleCVCB (the CVCB)</li><li>□ sampleMapBuff (the mapping buffer)</li></ul>
setupTPData	(copied from the background script of the routine card)
setupCVData	(copied from the background script of the routine card)
log	called by routine handlers to log the fact that they were called
logerror	called by conversation handlers to log an error message if there is an error
routine handlers	handlers for each of the routines used (copied from the routine card scripts), including <ul style="list-style-type: none"><li>□ TPAttach</li><li>□ TPDetach</li><li>□ MCAAllocate</li><li>□ MCDeallocate</li><li>□ MCSendData</li><li>□ MCRceiveAndWait</li><li>□ MCPPrepareToReceive</li></ul>
conversation handlers	handlers used to establish a conversation between Postmaster and Mailbox, send and receive data, and deallocate a conversation. <ul style="list-style-type: none"><li>□ closeConvLocal calls the MCDeallocate (kLocalDealloc) and TPDetach handlers after the other TP has deallocated the conversation</li><li>□ closeConv calls MCDeallocate (kSyncDealloc) and TPDetach (kNormalDetach) handlers to close the conversation (done while in the send state)</li><li>□ localAttach calls the setupTPData handler and then TPAttach (kLUAttach)—similar to the execTPAttach handler from the TPAttach card, but less general</li></ul>



<input type="checkbox"/> waitConv	calls the setupTPData handler and then TPAttach (kWaitAttach)—similar to the execTPAttach handler from the TPAttach card, but less general
<input type="checkbox"/> waitResult	waits for the async TPAttach (kWaitAttach) handler to finish by continually checking the ioResult
<input type="checkbox"/> detach	calls the TPDetach (kNormalDetach) handler
<input type="checkbox"/> receiveData	calls the MCRceiveAndWait handler
<input type="checkbox"/> sendData	calls the MCSendData handler
<input type="checkbox"/> prepareToReceive	calls the MCPprepareToReceive handler

### Postmaster card script

The Postmaster card script contains the openConvRemote handler that calls the waitConv handler to wait for the Mailbox to start a conversation. (The other handlers in the Postmaster card script are specific to the Postmaster application. They may be instructive, but it is not likely you will be able to use any of that code in your applications.)

### Mailbox card script

The Mailbox card script contains the openConv handler that calls the localAttach handler to establish a conversation with the Postmaster server. (The other handlers in the Mailbox card script are specific to the Mailbox application. They may be instructive, but it is not likely you will be able to use any of that code in your applications.)



---

---

## XData XCMDs and XFCNs

The XData XCMDs and XFCNs provide the interface between HyperCard APPC and MacAPPC. Specifically, the XData XCMDs and XFCNs define and use external data structures that are used to pass data to, and receive data from, MacAPPC routines. The data structures correspond in type and length to the partner program's data structures, or to the data structures used to pass parameters to APPC verbs. (The data structures are similar to the Pascal `RECORD` and the C `struct`.) You can use XData XCMDs and XFCNs to pass data between HyperCard and other applications.

There are four basic XData XCMDs and XFCNs:

- ☐ `xDefine` defines a data structure
- ☐ `xGlobal` allocates memory for a record
- ☐ `xPut` assigns values to the records fields
- ☐ `xGet` retrieves values from the records fields

These XCMDs and XFCNs automatically and transparently convert data between numeric and HyperTalk strings, as well as between ASCII and EBCDIC.

Additional XCMDs and XFCNs perform other necessary functions:

- ☐ `xLock` locks or unlocks a record
- ☐ `xPtr` returns a pointer to a locked record or a field of a locked record
- ☐ `xFill` fills a block of memory with a byte value
- ☐ `xSize` determines the size of a record or field (also determines whether or not a record has been defined—it returns 0 if not defined)
- ☐ `xDispose` frees memory allocated to a record or records
- ☐ `xMove` moves a block of memory
- ☐ `xResource` loads a code or data resource

The following sections describe the XData XCMDs and XFCNs in detail. They are listed approximately in the order that you would use them in a script. Parameters that appear in italics indicate that a string or numeric value should be substituted for that parameter as described. (The *fieldSpec* parameter is a special case. See "Special Parameters," following the commands.) Parameters not italicized are XData XCMD keywords that you should type literally. None of the command names or parameters are case sensitive. Square brackets—`[ ]`—indicate optional parameters. A vertical bar (`|`) means you should choose only one of the alternatives.



---

## xDefine

### Syntax

```
xDefine recordType [,AlignWord|AlignLong]

xDefine fieldName, fieldType [, arraySize]
:
:
xDefine endDef
```

### Description

`xDefine` defines a record type and specifies the names and types of the fields of the record type.

The first form of the `xDefine` statement names the record type (*recordType*).

By default, `xDefine` creates fields that align on a byte boundary. However, you can use one of the optional keywords (`AlignWord` or `AlignLong`) to align fields on a 16- or 32-bit boundary. See “Byte Alignment of Fields” at the end of this appendix.

After naming the record type, you must use the second form of the `xDefine` statement once for each field in the record. *fieldName* names the field, and *fieldType* specifies the field's type. *fieldType* can be one of the basic field types described in “Basic Data Types,” later in this appendix, or can itself be a record type defined by a previously executed `xDefine` command. The optional argument at the end of the statement (*arraySize*) lets you specify the size of the array or, if the field type is a string, the length of the string. Subsequent statements can refer to the elements of the array with index values ranging from 0 to *arraySize* – 1. See “xPut.”

End the series of field definitions with `xDefine endDef`.



---

## xGlobal

### Syntax

`xGlobal recordName [, recordType] [, size]`

### Description

`xGlobal` opens a record for use by the `xPut` and `xGet` commands. The *recordType* parameter is optional only if the named record has already been created by a previous `xGlobal` statement. Otherwise you must supply the *recordType* parameter, and the `xGlobal` statement creates (allocates storage for) a record of type *recordType*.

The optional *size* parameter lets you override the default size of the allocated record.

---

### Warning

If you specify a size in a subsequent `xGlobal` statement that is different from the previously specified size, the size of the handle to the record will be reset. Note that the statement may fail altogether if the record is locked.

---

Subsequent `xPut` and `xGet` statements can store values in and retrieve values from the fields of the record.

`xGlobal` returns a handle to the record in the `result` (the HyperCard container). You can use `xGlobal` in conjunction with other XCMDs and XFCNs that take a handle or a pointer as an argument.



---

## xPut

### Syntax

`xPut value, [typeCoercion, [length,]] fieldSpec`

### Description

`xPut` stores the value of a HyperTalk container or constant (*value*) in the field specified by *fieldSpec* of the record most recently opened by an `xGlobal` statement. (The field must be defined in that record.) The field specification names a field of the record, but does not include the record name.

See "Special Parameters" later in this appendix for a description of *fieldSpec*.

The field specification may be preceded by a type coercion (*typeCoercion*) prefix that temporarily forces the type of the field to a new type. If the new type is a string, then the prefix must include an additional parameter (*length*) specifying the length of the string.

*typeCoercion* has the form

`"#newTypeName"`

where `#newTypeName` is any one of the basic field types.

### Note

If *value* is the name of a MacAPPC constant, and *fieldSpec* specifies a numeric or Boolean field, then `xPut` places the value of the MacAPPC constant into the field, rather than the constant name. (See Appendix A for a list of MacAPPC constants.) This allows you to use `xPut` with a MacAPPC constant name without looking up the constant's value with `xConst`.

In the example below, both `xPut` lines place the value 2 (the value of the `kWaitAttach` constant) into the `tpAttachType` field of the `tpData` record. The example assumes that you defined `APPCTPRec` (using `xDefine`) as it is defined by the `defineCVandTP` handler in the script of the first card of the HyperCard APPC stack.

```
xGlobal "tpData", "APPCTPRec"
xPut "kWaitAttach", "tpAttachType"
xPut xConst("kWaitAttach"), "tpAttachType"
```



---

## xGet

- Syntax** `xGet (fieldSpec)`
- Description** `xGet` is a function that returns the value stored in the field specified by *fieldSpec*. (See “Special Parameters,” later in this appendix, for information about the *fieldSpec* parameter. See “xPut” for information about type coercion.) If the type of *fieldSpec* is a string, then `xGet` will always return an ASCII string without the length byte or length word, and without space padding.

---

## xLock

- Syntax** `xLock recordName, on|off`
- Description** `xLock` locks (on) or unlocks (off) the specified record (*recordName*). You must lock a record while using the `xPtr` function so that the record does not move in memory while pointers to fields in the record are being accessed. (See “xPtr.”)

---

## xPtr

- Syntax** `xPtr (recordName [, fieldSpec])`
- Description** `xPtr` is a function that returns a pointer to the specified record (*recordName*), or to a field within that record (*fieldSpec*). (See “Special Parameters,” later in this appendix, for information about the *fieldSpec* parameter.) The record must have been locked by a previous `xLock` statement. (See “xLock.”) The returned pointer value remains valid only as long as the record is locked.
- Note** You should unlock the record with the `xLock` command when the pointer value is no longer needed.



---

## **xFill**

- Syntax** `xFill value, recordName [, fieldSpec]`
- Description** `xFill` fills the specified field (*fieldSpec*) of the specified record (*recordName*) with the numeric value of the HyperTalk constant or container (*value*). (See "Special Parameters," later in this appendix, for information about the *fieldSpec* parameter.) If you do not specify a field, then `xFill` fills the entire record. *value* must be between 0 and 255. The most common values are 0 (for NULL fields and NIL pointers), 32 (ASCII space), and 64 (EBCDIC space).
- Note** Because the value is numeric, there is no conversion between ASCII and EBCDIC.

---

## **xSize**

- Syntax** `xSize(recordName [, fieldSpec])`
- Description** `xSize` is a function that returns the size of the specified record (*recordName*) or the size of the specified field within that record (*fieldSpec*). (See "Special Parameters," later in this appendix, for information about the *fieldSpec* parameter.) You can use `xSize` to determine if a record exists: `xSize` returns 0 if the specified record does not exist.

---

## **xDispose**

- Syntax** `xDispose [recordName]`
- Description** `xDispose` frees the storage allocated to the specified record (*recordName*) or to all records allocated by `xGlobal` statements if no record is named.



---

## xMove

<b>Syntax</b>	<code>xMove sourcePtr, destinationPtr, length</code>
<b>Description</b>	<p><code>xMove</code> moves the number of bytes specified in <i>length</i> from the location pointed to by <i>sourcePtr</i> to the location pointed to by <i>destinationPtr</i>.</p> <p><i>sourcePtr</i> and <i>destinationPtr</i> should be values returned by the <code>xPtr</code> function; <i>length</i> should be a value returned by the <code>xSize</code> function. You can, however, use any valid pointer values and any length. Use with care and skill.</p>
<b>Note</b>	The <code>xMove</code> command calls the User Interface Toolbox procedure <code>BlockMove</code> . See <i>Inside Macintosh</i> , Volume II, for more information.

---

## xResource

<b>Syntax</b>	<code>xResource subCommand, parameterList</code>
<b>Description</b>	<p><code>xResource</code> loads a code resource to be used as a mapper procedure for a mapped conversation or as an I/O completion routine. You can also use <code>xResource</code> to load resource data to be sent to a partner application in a MacAPPC conversation.</p> <p><code>xResource</code> has several subcommands (<i>subCommand</i>). They are listed below with their corresponding parameter lists (<i>parameterList</i>).</p> <ul style="list-style-type: none"><li>❖ <i>Note:</i> HyperTalk global variable names must be enclosed by double quotation marks so that HyperTalk passes the names and not the values to the <code>xResource</code> command.</li><li>■ <code>getLockedRes, resType, resName, "resHandle", "resPointer", "resSize"</code><p><code>xResource getLockedRes</code> loads and locks the resource named <i>resName</i> of type <i>resType</i>, and returns a handle to the resource in the HyperTalk global variable <code>resHandle</code> and a pointer to the resource in the HyperTalk global variable <code>resPointer</code>. If the resource was successfully loaded, <code>xResource getLockedRes</code> sets the HyperCard global variable <code>resSize</code> to the size of the resource.</p></li><li>■ <code>getRes, resType, resName, "resHandle", "resSize"</code><p><code>xResource getRes</code> loads the resource named <i>resName</i> of type <i>resType</i> and returns a handle to the resource in the HyperTalk global variable <code>resHandle</code>. If the resource was successfully loaded, <code>xResource getRes</code> sets the HyperCard global variable <code>resSize</code> to the size of the resource.</p></li></ul>



- `getResData, resType, resName, xGlobalName, resSize`

`xResource getResData` loads a resource named *resName* of type *resType* into a previously declared `xGlobal` named *xGlobalName*. If the resource was successfully loaded, `xResource getResData` sets the HyperCard global variable *resSize* to the size of the resource.

After using `xResource getResData`, a subsequent call to `xGlobal` using *xGlobalName* will return the handle to the resource, which can no longer be purged. You can use `xLock` and `xPtr` to lock and obtain a pointer to the resource data. You can use `xDispose` to dispose of the resource handle.

- `lock, resHandle, "resPointer"`

`xResource lock` locks the resource specified by *resHandle* and returns a pointer to the locked resource in the HyperTalk global variable *resPointer*.

- `unlock, resHandle`

`xResource unlock` unlocks the resource specified by *resHandle*.

- `release, resHandle`

`xResource release` releases the memory occupied by the resource specified by *resHandle*. After using `xResource release`, you should not refer to *resHandle* until the resource is reloaded with `xResource getRes` or `xResource getLockedRes`.



---

## Special parameters

The following sections detail the syntax of the *fieldSpec* and *ptrType* parameters. *fieldSpec* is a special parameter—used by several XData XCMDs and XFCNs—that describes a record field. It is actually a set of parameters, and has several variations, depending on the type of record. One variation of *fieldSpec* uses a pointer field, described by *ptrType*. *ptrType* is, itself, a set of parameters with several variations. See “XData XCMDs and XFCNs” for an explanation of the syntax symbols, and descriptions of the XData XCMDs and XFCNs that use *fieldSpec*. See the second handler in “Sample Handlers” for examples of how to use XData XCMDs and XFCNs with the *fieldSpec* and *ptrType* parameters.

### *fieldSpec* parameter

The following XData XCMDs and XFCNs use the *fieldSpec* parameter:

- xPut
- xGet
- xPtr
- xFill
- xSize

*fieldSpec* has three variations. Note that the first form of *fieldSpec* is recursive.

- *recordField*, *fieldSpec*

where

*recordField* is a field of a record defined by xDefine

- *basicField* [, *arrayIndex*]

where

*basicField* is a field of a basic data type (See “Basic Data Types,” later in this chapter.)

*arrayIndex* is an integer index into the array (when the specified field is an array)

- *ptrField*, *ptrType*

where

*ptrField* is a field whose basic type is PTR

*ptrType* (See the next section, “*ptrType* Parameter.”)



## ***ptrType* parameter**

The *ptrType* parameter has three variations. Note that the first two forms of *ptrType* are recursive.

□ "PTR", *ptrType*

□ *recordType*, *fieldSpec*

where

*recordType* is a record type defined by *xDefine*

*fieldSpec* (See the previous section, "*fieldSpec* Parameter.")

□ *basicType*

where

*basicType* is one of the predefined basic types (See "Basic Data Types," later in this chapter.)

---

## **Sample handlers**

The following handler gives an example of how to use the XData XCMDs and XFCNs.

```
on mouseUp
  global hMapper,pMapper  -- handle and pointer to mapping procedure
  global size -- size of mapping procedure

  -- Define a record data type called CustomerRec
  xDefine CustomerRec      -- create the record
  xDefine Name,EUString,20 -- add a string field called "Name"
  xDefine NickName,EUString,20 -- add a string field called "NickName"
  xDefine CustomerID,INTEGER -- add an integer field called
                             -- "CustomerID"
  xDefine endDef          -- end the definition of "CustomerRec"

  -- Allocate space for "Customer1" which is of type "CustomerRec"
  xGlobal CustomerData1,CustomerRec

  -- Allocate space for "Customer2" which is also of type "CustomerRec".
  -- Subsequent xPuts and xGets will refer to this record until another
  -- xGlobal is executed to open up a different record.
  xGlobal CustomerData2,CustomerRec

  -- Put "Smith" converted to EBCDIC uppercase and blank padded into name
  -- field
  xPut "Smith",Name

  -- Put the name (converted back to ASCII) into a hypercard variable
  -- theName
  put xGet(Name) into theName

  put xSize(CustomerData2,Name) into size
    -- size = sizeof(customerData2.Name)

  -- Copy the "Name" field of CustomerData2 to the "NickName" field of
  -- CustomerData1
  xLock CustomerData1,on -- lock the record so it doesn't move in memory
  xLock CustomerData2,on -- lock the record so it doesn't move in memory
  xMove xPtr(CustomerData2,Name),xPtr(CustomerData1,NickName),size
  xLock CustomerData1,off -- unlock the record
  xLock CustomerData2,off -- unlock the record
```



```

xFill 64, CustomerData1, Name      -- fill the name field with EBCDIC spaces
xGlobal CustomerData1      -- Open up the CustomerData1 record
put xGet(NickName) into theName -- put customer name into "theName"
xPut theName, Name -- put contents of "theName" into the
    -- "Name" field
put xSize(CustomerData2) -- "42" will appear in the msg box
put xGet(Name) after msg -- "Smith" will appear in the msg box

xDispose CustomerData1 -- dispose the storage allocated for
CustomerData1
xDispose CustomerData2 -- dispose the storage allocated for
CustomerData2

-- Load and lock the mapping procedure resource called "SampleMapper"
-- which is of type PMAP. Note that the handle, pointer, and size
-- parameters must all be HyperCard globals.
xResource GetLockedRes, "PMAP", "SampleMapper", "hMapper", "pMapper", "size"

xResource Unlock, "hmapper"      -- unlock the mapping procedure
xResource Release, "hmapper"     -- release the mapping procedure
end mouseUp

```

The next, more advanced handler, demonstrates how to use XData XCMDs and XFCNs with pointer and record fields.

```

on mouseUp
-- Create a record type called "NameRec" with 2 string fields
xDefine NameRec
xDefine  FirstName, STRING, 20
xDefine  LastName,  STRING, 20
xDefine endDef

-- Create a record type called "EmployeeRec" with 3 fields.
-- The "Name" field is simply a field of type "NameRec".
-- The "NamePtr" field is of type PTR and will eventually
-- point to the "Name" field. The "NameHdl" field is also
-- of type PTR. It will eventually point to the "NamePtr"
-- field which will make it a handle to the "Name" field.
xDefine EmployeeRec
xDefine  Name, NameRec
xDefine  NamePtr, PTR
xDefine  NameHdl, PTR
xDefine endDef

-- Allocate the "employee" record.
xGlobal employee, EmployeeRec

xPut "John", Name, FirstName -- Set "employee.Name.FirstName"
xPut "Doe", Name, LastName  -- Set "employee.Name.LastName"

xLock employee, on -- Need to lock the record before you can use xPtr
xPut xPtr(employee, name), NamePtr -- Set "NamePtr"
xPut xPtr(employee, NamePtr), NameHdl -- Set "NameHdl"

-- The following are all ways of accessing the
-- employee's last name.
put xGet(Name, LastName) into msg
put xGet(NamePtr, NameRec, LastName) after msg
put xGet(NameHdl, PTR, NameRec, LastName) after msg

-- The following are all ways of accessing the
-- employee's first name.
put xGet(Name, FirstName) after msg

```



```

put xGet(NamePtr, STRING, 20) after msg
put xGet(NamePtr, NameRec, FirstName) after msg
put xGet(NameHdl, PTR, NameRec, FirstName) after msg

xLock employee, off -- unlock the employee record
xDispose employee -- dispose of the employee record
end mouseUp

```

---

## XData errors

If you use a HyperCard APPC XCMD incorrectly, HyperCard will display an alert stating

"Can't understand ..."

and open the script and place the insertion point at the line that caused the error. The HyperCard global variable `xDataErr` will contain an error code defining the problem. Table H-5 shows the possible error codes and their meanings.

**Table H-4**  
XData errors

Code	Description
0	No error found
10000	xDefine field length is too large
10001	Too few arguments passed to XCMD
10002	Data type not found (undefined)
10003	Incorrect syntax used
10004	Incorrect use of xDefine endDef
10005	No xGlobal records have been defined
10006	xGlobal record not defined
10007	Invalid array index
10008	Field not found
10009	Type already exists
10010	Field already exists
10011	Not currently defining a record
10012	Record not locked
10013	Record already locked
10014	Illegal parameter given
10015	PICT resource not found
10016	Constant not found
10017	Specified type is not a basic type
10019	Memory allocation error
10020	Call to SetHandleSize failed
10021	NIL or 0 passed to get62srvr XCMD



## Basic data types

Table H-6 describes the basic MacAPPC data types. The first column contains the keyword to be used in place of *fieldType* in the `xDefine` command. (See “xDefine” earlier in this appendix.) The Size column specifies the maximum size of the field, including the length byte or length word.

**Table H-5**  
Basic data types

Type name	Size	Description
BYTE	1 byte	Unsigned
BOOLEAN	1 byte	TRUE, FALSE, 1, 0
INTEGER	2 bytes	Signed
LONGINT	4 bytes	Signed
PTR	4 bytes	Pointer
CHAR	1 byte	ASCII character
ECHAR	1 byte	EBCDIC character
PSTRING	<= 256 bytes*	ASCII string with length byte
CSTRING	<= 256 bytes*	ASCII string null terminated
STRING	<= 32,768 bytes*	ASCII string padded with spaces
ESTRING	<= 256 bytes*	EBCDIC string padded with spaces
EUSTRING	<= 256 bytes*	EBCDIC string padded with spaces
LSTRING	<= 32,768 bytes*	ASCII string with length word
ELSTRING	<= 32,768 bytes*	EBCDIC string with length word
EULSTRING	<= 32,768 bytes*	EBCDIC string with length word

\* Maximum size including length byte(s) or null terminator.

For PSTRING, CSTRING, and LSTRING, the maximum lengths specified by the *arraySize* parameter of `xDefine` and the *length* parameter of `xPut`, are as follows:

PSTRING	maximum length of a Pascal string <i>not including</i> the length byte
CSTRING	maximum length of a C string (null terminated) <i>including</i> the terminating null character
LSTRING	maximum length of a string <i>not including</i> the length word

The data types beginning with *E* provide conversion between ASCII and EBCDIC, for use when data is to be exchanged with IBM hosts that use the EBCDIC character set. Fields of type ESTRING and EUSTRING store data as EBCDIC characters, padded to the specified length with spaces. The `xPut` command converts a string from ASCII to EBCDIC. The `xGet` function converts the stored string from EBCDIC to ASCII and returns the ASCII result.



Fields of type `EUSTRING` and `EULSTRING` store data as EBCDIC characters in all uppercase. The `xPut` command converts a string from mixed uppercase and lowercase ASCII to uppercase EBCDIC. The `xGet` function does not do any case conversion.

HyperCard always stores strings in ASCII. The conversion between ASCII and EBCDIC is automatic and transparent.

### **Byte alignment of fields**

When allocating storage for a record, the `xDefine` command aligns each field on a byte boundary. Additional space can be created by inserting dummy fields, or by using the optional `AlignWord` keyword or `AlignLong` keyword in the `xDefine` statement. (See “`xDefine`” earlier in this appendix.) `AlignWord` causes each field that is longer than 1 byte in length to align on a 16-bit boundary. `AlignLong` causes each field that is longer than 1 byte in length to align on a 32-bit boundary.







## Appendix I

### MacAPPC Option Sets

The following table shows the APPC option sets supported by MacAPPC at the time of publication.

**Table I-1**  
Supported APPC option sets

Option Sets	Supported	Not supported
Conversations between programs located at the same LU	x	
Delayed allocation of a session	x	
Immediate allocation of a session	x	
Sync point services		x
Session-level LU-LU verification	x	
User ID verification	x	
Program-supplied user ID and password	x	
User ID authorization	x	
Profile verification and authorization	x	
Profile passthrough	x	
Program supplied profile	x	
PIP data	x	
Logging of data in a system log	x	
Flush the LU's send buffer	x	
LUW ID	x	
Prepare-to-receive	x	
Long locks	x	
Post on receipt with wait	x	
Post on receipt with test for posting	x	
Receive immediate	x	
Test for request-to-send received	x	



**Table I-1 (continued)**  
Supported APPC option sets

Option Sets	Supported	Not supported
Data mapping	x	
Function Management Header (FMH) data	x	
Get attributes	x	
Get conversation type	x	
Mapped conversation LU services component	x	
CHANGE SESSION LIMIT verb	x	
MIN COWINNERS TARGET parameter	x	
RESPONSIBLE (TARGET) parameter	x	
DRAIN TARGET (NO) parameter	x	
FORCE parameter	x	
ACTIVATE SESSION verb	x	
DEACTIVATE SESSION verb	x	
LU-parameter verbs	x	
LU-LU session limit	x	
Locally-known LU names	x	
Uninterpreted LU names	x	
Single-session reinitiation	x	
Maximum RU size bounds	x	
Session-level mandatory cryptography		x
Contention winner automatic activation limit	x	



## Appendix J

### ASCII/EBCDIC Tables

The following tables provide the translation from ASCII to EBCDIC and vice versa. In both tables, you read the first hexadecimal digit for the character from the vertical row of digits and the second hexadecimal digit from the horizontal row.

**Table J-1**  
ASCII to EBCDIC

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
1	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2	40	4F	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	4A	E0	5A	5F	6D
6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	6A	D0	A1	07
8	20	21	22	23	24	15	06	17	28	29	2A	2B	2C	09	0A	1B
9	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	E1
A	41	42	43	44	45	46	47	48	49	51	52	53	54	55	56	57
B	58	59	62	63	64	65	66	67	68	69	70	71	72	73	74	75
C	76	77	78	80	8A	8B	8C	8D	8E	8F	90	9A	9B	9C	9D	9E
D	9F	A0	AA	AB	AC	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7
E	B8	B9	BA	BB	BC	BD	BE	BF	CA	CB	CC	CD	CE	CF	DA	DB
F	DC	DD	DE	DF	EA	EB	EC	ED	EE	EF	FA	FB	FC	FD	FE	FF



**Table J-2**  
EBCDIC to ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
1	10	11	12	13	9D	85	08	87	18	19	92	8F	1C	1D	1E	1F
2	80	81	82	83	84	0A	17	1B	88	89	8A	8B	8C	05	06	07
3	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4	20	A0	A1	A2	A3	A4	A5	A6	A7	A8	5B	2E	3C	28	2B	5D
5	26	A9	AA	AB	AC	AD	AE	AF	B0	B1	21	24	2A	29	3B	5E
6	2D	2F	B2	B3	B4	B5	B6	B7	B8	B9	7C	2C	25	5F	3E	3F
7	BA	BB	BC	BD	BE	BF	C0	C1	C2	60	3A	23	40	27	3D	22
8	C3	61	62	63	64	65	66	67	68	69	C4	C5	C6	C7	C8	C9
9	CA	6A	6B	6C	6D	6E	6F	70	71	72	CB	CC	CD	CE	CF	D0
A	D1	7E	73	74	75	76	77	78	79	7A	D2	D3	D4	D5	D6	D7
B	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7
C	7B	41	42	43	44	45	46	47	48	49	E8	E9	EA	EB	EC	ED
D	7D	4A	4B	4C	4D	4E	4F	50	51	52	EE	EF	F0	F1	F2	F3
E	5C	9F	53	54	55	56	57	58	59	5A	F4	F5	F6	F7	F8	F9
F	30	31	32	33	34	35	36	37	38	39	FA	FB	FC	FD	FE	FF





## Appendix K



### Configuration Worksheets

The worksheets in this appendix are intended to help you use the MacAPPC Configuration program described in Chapter 11 of this guide. For a remote configuration, use the worksheets in both Figure K-1 and Figure K-2. For a local configuration, use the worksheet in Figure K-2 only.

In the worksheets, the default values are placed outside of the screens themselves to provide those values for quick reference but still allow you to write in the correct value on the worksheet. The shaded fields represent information that MacAPPC will fill in based upon earlier entries.



# MacAPPC™ Configuration Worksheet

**Node**

Exchange ID:

Access Type: ☒ SDLC

Monitor Timer:

**Multipoint configuration**

Multipoint Primary

**Line**

Line Name:

Line Type: SDLC

---

Line Number: ☐ 1 ☐ 2 ☐ 3 ☐ 4

Role Type: ☐ Prim ☐ Sec ☐ Negot

Connect Type: ☐ Lease ☐ Multi ☐ Switch

Max BTU:

Line Speed: ☐ 300 ☐ 1200 ☐ 2400

☐ 4800 ☐ 9600 ☐ 19200

Max Retries:

Idle Time:

NP Recv Time:

Max I-Frames:

NRZI Support: ☐ NRZ ☐ NRZI

Duplex Type: ☐ Half ☐ Full

1

Negot

Lease

265

9600

3

800

10000

7

NRZ

Half

Multipoint Secondary

**Line**

Line Name:

Line Type: SDLC

---

Line Number: ☐ 1 ☐ 2 ☐ 3 ☐ 4

Role Type: ☐ Prim ☐ Sec ☐ Negot

Connect Type: ☐ Lease ☐ Multi ☐ Switch

Max BTU:

Line Speed: ☐ 300 ☐ 1200 ☐ 2400

☐ 4800 ☐ 9600 ☐ 19200

Max Retries:

Idle Time:

NP Recv Time:

Max I-Frames:

NRZI Support: ☐ NRZ ☐ NRZI

Duplex Type: ☐ Half ☐ Full

1

Negot

Lease

265

9600

3

800

10000

7

NRZ

Half

FFF00001

30

**Point-to-point configuration**

**Partner**

Partner Name:

Line Name:

☐ Exch ID:

☐ CPU ID:

ALS Address:

Phone Number:

Exch ID: FFF00000

C1

**Partner**

Partner Name:

Line Name:

☐ Exch ID:

☐ CPU ID:

ALS Address:

Phone Number:

Exch ID: FFF00000

C1

**Figure K-1**  
MacAPPC remote configuration worksheet

The callouts in the margin are defaults

In a Multipoint Secondary configuration, this address matches the ALS Address defined in the primary partner field



# MacAPPC™ Configuration Worksheet

**Local LU**

Local LU:

LU ID:

Net Name:

Net Qual:

Max Sessions:

LU Security: ☐ Yes ☐ No

Wait Time:

Max TPs:

User IDs:

Profiles:

Password:

1

1

No

60

5

1

1

1

Yes

No

**TP**

TP Name:

Local LU:

Net Name:

Status: ☐ Enable ☐ Temp ☐ Perm

Conv Type: ☐ Basic ☐ Map ☐ Either

Sync Level: ☐ None ☐ Confirm

PIP: ☐ Yes ☐ No

PIP Count:

PIP Check: ☐ Yes ☐ No

Data Mapping: ☐ Yes ☐ No

FMH Data: ☐ Yes ☐ No

Privilege: ☐ None ☐ CNOS ☐ Session

LUV: ☐ Yes ☐ No

Security Required: ☐ None ☐ Conv ☐ User

☐ Prof ☐ User/Prof

User ID:

Profile:

**TP**

TP Name:

Local LU:

Net Name:

Status: ☐ Enable ☐ Temp ☐ Perm

Conv Type: ☐ Basic ☐ Map ☐ Either

Sync Level: ☐ None ☐ Confirm

PIP: ☐ Yes ☐ No

PIP Count:

PIP Check: ☐ Yes ☐ No

Data Mapping: ☐ Yes ☐ No

FMH Data: ☐ Yes ☐ No

Privilege: ☐ None ☐ CNOS ☐ Session

LUV: ☐ Yes ☐ No

Security Required: ☐ None ☐ Conv ☐ User

☐ Prof ☐ User/Prof

User ID:

Profile:

**TP**

TP Name:

Local LU:

Net Name:

Status: ☐ Enable ☐ Temp ☐ Perm

Conv Type: ☐ Basic ☐ Map ☐ Either

Sync Level: ☐ None ☐ Confirm

PIP: ☐ Yes ☐ No

PIP Count:

PIP Check: ☐ Yes ☐ No

Data Mapping: ☐ Yes ☐ No

FMH Data: ☐ Yes ☐ No

Privilege: ☐ None ☐ CNOS ☐ Session

LUV: ☐ Yes ☐ No

Security Required: ☐ None ☐ Conv ☐ User

☐ Prof ☐ User/Prof

User ID:

Profile:

Enable

Fither

None Confirm

No

No

No

None

No

**Mode**

Mode Name:

Local LU:

Remote LU:

Adj Station:

Send Pacing:

Recv Pacing:

Max RU UB:

Max RU LB:

Sync Level: ☒ Confirm

Sess Reinit: ☐ None ☐ Oper ☐ Prim

☐ Sec ☐ Either

Max Sessions:

Min 1st Spkrs:

PB Sessions:

Queue Binds: ☐ Yes ☐ No

Blank Mode: ☐ Yes ☐ No

None

1

1

1

Yes

No

3

3

1021

256

**Remote LU**

Remote LU:

Local LU:

Net Name:

Net Qual:

CP Name:

Init Q Req: ☐ Yes ☐ No

Parallel Sess: ☐ Yes ☐ No

Remote LU box feeds Remote LU Name into Mode

Figure K-2  
MacAPPC local configuration worksheet



1

2

3





## Glossary

**abend:** A condition of a transaction program that is the result of an error in execution. (The word is a contraction of the terms *abnormal* and *end*).

**ACF/VTAM:** See **Virtual Telecommunications Access Method**.

**adjacent link station (ALS):** The hardware and software within an SNA node that control the link connection and prepare for communication transmissions.

**Adjacent Link Station Control Block:** A block of memory allocated and maintained by an adjacent link station to maintain state and control information about an individual connection to a MacAPPC server.

**ADSP:** See **AppleTalk Data Stream Protocol**.

**Advanced Program-to-Program Communication (APPC):** An enhancement to SNA that permits nodes on a network to communicate as equals (peers), in contrast to the traditional master-slave relationship that exists in a mainframe-dominated network. Sometimes called *Advanced Peer-to-Peer Communication*.

**allocated:** A conversation is allocated when one partner program requests that the other be attached at the appropriate remote LU.

**allocate routine:** A routine that identifies the desired type of session by specifying a mode name, and identifies the target program by specifying its name and the name of the remote LU at which the target program is located.

**API:** See **Application Programming Interface**.

**APPC:** See **Advanced Program-to-Program Communication**.

**APPC Mail:** The sample HyperCard APPC application provided with the HyperCard APPC stack. APPC Mail is actually two applications in one: Postmaster and Mailbox.

**Apple IPC:** A component of the MCP operating system that provides inter-process communication (IPC) services to programs or processes on the Macintosh II computer that communicate with processes on one or more communication cards.

**AppleTalk Data Stream Protocol (ADSP):** A symmetric, connection-oriented protocol for establishment and maintenance of full-duplex streams of data bytes between two sockets in an AppleTalk internet.

**AppleTalk Manager:** An interface to a set of device drivers that enable programs to send and receive information via an AppleTalk network.

**AppleTalk network system:** The system of network software and hardware used in various implementations of Apple's communications network.

**Application Programming Interface:** The handlers, XCMDs, and XFNCs that provide the interface between HyperCard and MacAPPC.

**application subsystem:** A type of SNA program that contains part of the code that implements LUs and provides an interface to other SNA products. Includes transaction processing systems, such as CICS/VS, and interactive support systems, such as time share option (TSO).

**ASCII:** Acronym for *American Standard Code for Information Interchange* (pronounced ASK-ee). A standard that assigns a unique binary number to each text character and control character. Compare **EBCDIC**.

**asynchronous:** (adj.) Able to continue processing while the original request is still executing. When an application makes an asynchronous MacAPPC call, control is returned immediately to the calling program. Compare **synchronous**.



**attach:** The connection between a transaction program and a MacAPPC server. The attach must be established before a conversation can take place.

**basic conversation:** A conversation between TPs using basic conversation routines. Typically, service transaction programs use basic conversations.

**basic conversation routine:** A type of MacAPPC routine that allows a programmer maximum flexibility and control over the conversation. Compare **mapped conversation routines**.

**basic transmission unit (BTU):** An SNA message unit that includes a request/response header and a transmission header.

**bidding:** The initiation of a bracket or a conversation.

**bind:** A request that is sent from a primary LU to a secondary LU to activate an LU-LU session.

**bracket:** (n.) A time-slice of the session in which each serial session conversation is mapped. (v.) To keep together a sequence of related message units that flow between LUs and constitute a single logical-unit-of-work.

**BTU:** See **basic transmission unit**.

**buffer:** Temporary data storage used by the LUs at each end of a conversation to assist with sending and receiving data.

**change-number-of-sessions (CNOS):** (1) A type of control operator routine that permits modification of sessions between LUs. (2) A service transaction program that issues CNOS routines and negotiates with a control operator transaction program to set session limits.

**CICS:** See **Customer Information Control System**.

**client:** A computer that has access to services on a network. The computers that provide services are called *servers*. A user at a client may request file access, remote log-on, file transfer, printing, or other available services from servers.

**CNOS:** See **change-number-of-sessions**.

**CNOS routines:** Routines that change the LU or mode session limit, which controls the number of sessions per mode name that are available between two LUs for allocation to conversations.

**communication controller:** A type of communication control unit that has its operations controlled by a program stored and executed at that unit. For example, the IBM 3704 and 3705 are communication controllers.

**component:** A key part of an LU 6.2 network. Components include local nodes, local LUs, TPs, lines, partners (consisting of stations and control points), remote LUs, and modes. A session is not a component. Components can be logical (for example a local LU) or physical (for example a line).

**configuration:** A specific instance of a MacAPPC network.

**configuration file:** A Macintosh resource file created by the Configuration program that describes the components making up an instance of a MacAPPC network.

**configuring:** The process of defining network components using MacAPPC definition routines. The Administration program does this immediately after downloading the server. The combination of these two steps is known as *starting the server*.

**confirm:** One level of resource synchronization that helps to ensure consistency within a distributed transaction. The LU's only involvement is to communicate the confirmation request and reply between the programs. Any recovery action is conducted at the program level.

**connectivity:** The presence of compatible communication protocols.

**control block:** A block of memory allocated by a program and used to maintain control information. Several types of control blocks are used by MacAPPC. See **Transaction Program Control Block**, **Conversation Control Block**, **Adjacent Link Station Control Block**, and **Partner Control Block**.

**control operator:** A set of MacAPPC routines that control aspects of an LU and its configuration elements.

**control point:** The node at which control information is added and removed by the functional layers of SNA.



**Control Point Block:** A block of memory allocated by the control point for use by MacAPPC drivers to maintain state and control information about an individual connection to a MacAPPC server.

**conversation:** A logical connection between two transaction programs over a session.

**Conversation Control Block (CVCB):** A block of memory used by MacAPPC device drivers to maintain state and control information about an individual conversation with a partner transaction program. The CVCB is allocated and maintained by a transaction program.

**conversation resource:** A resource by which one transaction program invokes (attaches) another, and by which the two programs exchange messages and state information.

**conversation states:** During a conversation, a transaction program is in one of several defined states. The conversation states are reset, send, receive, confirm, confirm/send, confirm/deallocate, and deallocate.

**Customer Information Control System (CICS):** A teleprocessing monitor for host computers. CICS is designed to reduce the effort required for terminal-oriented transaction programming by providing an interface between user-written applications and the telecommunications access method (such as VTAM) or database manager.

**data mapping:** The process of transforming data from the form used by a transaction program into a data stream that is sent to a remote LU.

**data stream:** The allowable characters, including control codes, that flow between two LUs.

**delta guide:** A description of something new in terms of its differences from something the reader already knows about. The name comes from the way mathematicians use the Greek letter delta ( $\Delta$ ) to represent a difference.

**Device Manager:** The part of the Macintosh Operating System that supports device input, output, and control.

**DIA:** See **Document Interchange Architecture**.

**DISOSS:** See **Distributed Office Support System**.

**Distributed Office Support System (DISOSS):** An IBM office automation product that uses DIA for document formatting.

**distributed transaction:** The cooperative execution of multiple programs at multiple locations to achieve some user-requested processing function.

**distributed transaction processing:** The capability of a network to allow user-application processing to be executed cooperatively on various interconnected, but physically separated, computing systems.

**Document Interchange Architecture (DIA):** An IBM architecture that is designed for exchanging messages and documents and that provides library services. DIA is implemented as a set of service transaction programs.

**EBCDIC:** Acronym for *Extended Binary-Coded Decimal Interchange Code* (pronounced "EB-si-dik"). A code used by IBM that represents each letter, number, special character, and control character as an 8-bit binary number. EBCDIC has a character set of 256 8-bit characters. Compare **ASCII**.

**end user services:** Services concerned with the exchange of data between two logical units such as are used in implementing LU-to-LU sessions.

**external command:** See **XCMD**.

**external function:** See **XFNC**.

**FAPL:** See *SNA Format and Protocol Reference Manual for LU Type 6.2*.

**file window:** A window displayed in the Configuration program when a file is opened or created.

**flush:** To force an LU to send the information in its send buffer.

**FMH:** See **Function Management Header**.

**Forwarder (FWD):** The ADSP Forwarder is a component of the MCP operating system that provides a client-server interface between services running under the MCP operating system and clients connected to the server computer via AppleTalk ADSP connections.

**frame:** A data unit used in data link services.



**Function Management Header (FMH):** A header on a request/response unit that is used as part of the end-user services for functions such as specifying a device to display data or controlling the way data is processed.

**FWD:** See **Forwarder**.

**General Data Stream (GDS):** A means of formatting information for transfer over an SNA network.

**handler:** A specific portion of a HyperTalk script that performs some action in HyperCard. Handlers begin with the command line "on *handlerName*" and conclude with the command line "end *handlerName*" where *handlerName* is the word that causes the handler to be executed.

**half-session layer:** The layer that controls session-level communication between LUs.

**hexadecimal characters:** The representation of numbers in the base-16 system, using the ten digits 0 through 9 and the six letters A through F.

**HyperCard APPC:** The HyperCard-based front-end for MacAPPC. The stack contains the HyperCard APPC Application Programming Interface (API), a "lab" for experimenting with MacAPPC routines, and a sample application (APPC Mail).

**I-frame:** One of three types of SDLC control field formats. This format consists of a receive count field, poll/final bit, send count field, and frame identifier.

**initiating a conversation:** The stepwise process of allocating a session for the conversation and attaching the target program.

**intelligent device:** A device that contains a microprocessor and a program that allows the device to interpret data sent to it as commands that the device is to perform.

**internet:** Any interconnected group of networks; for example, an interconnected group of AppleTalk network systems.

**Inter-Process Communication (IPC):** Communication between processes or tasks. MacAPPC uses Apple IPC to provide inter-process communication services to programs or processes on the Macintosh II computer that are used to communicate with processes on one or more intelligent communications cards.

**interprogram communication:** The set of services that allows programs using APPC to communicate with each other.

**IPC:** See **Inter-Process Communication**.

**jumpers:** A type of hardware switch made up of pins and removable jumper blocks. The switch can be set using the jumper block to connect specific pins.

**leased:** A directly connected line.

**line:** A network component that provides a physical link to other nodes.

**local LU:** The LU that is being used by a given transaction program in a conversation. The other LU is the remote LU.

**local resources:** Transaction program resources that are created locally (by the product, and independently of SNA). These include local files, databases, queues, logs, and terminals.

**logging:** The process of recording all internal server messages.

**logical unit:** A defined software component of SNA that provides a point of access, or port, to permit a user to gain access to an SNA network. LUs provide transmission capabilities and a set of services for the user. There are several varieties of LU, including LU type 6.2, which is the LU type that currently has the most comprehensive set of defined capabilities. MacAPPC uses LU type 6.2.

**logical-unit-of-work (LUW):** An identifier assigned by a transaction program for each transaction issued by the program.

**LU:** See **logical unit**.

**LU 6.2 network:** A particular type of SNA network that provides a connection between its transaction programs and network resources.

**LU services manager:** Provides management of the internal allocation of LU resources and invokes other functions of the node on behalf of end users.

**LUW:** See **logical-unit-of-work**.

**Macintosh Programmer's Workshop (MPW):** Apple's software development environment for the Macintosh family.

**Mailbox:** The part of APPC Mail that receives text and graphics from the Postmaster.



**mapped conversation:** A protocol boundary that allows application transaction programs to exchange data records of arbitrary format, irrespective of the data streams used by the underlying products.

**mapped conversation buffer:** A block of memory that holds data sent or received from a partner transaction program before the actual mapping of the data is performed. The block of memory is allocated and maintained by a transaction program.

**mapped conversation routine:** A type of conversation routine used by transaction programs for functions such as data mapping. Compare **basic conversation routine**.

**mapper:** A routine used by MacAPPC when send or receive data mapping is required to translate data and map names.

**master:** The central processor in a network. Compare **slave**.

**Memory Manager:** The part of the Macintosh Operating System that dynamically allocates and releases memory space in the heap.

**mode:** A single-session connection or a group of parallel sessions having similar session-level parameters and path-control characteristics.

**MPW:** See **Macintosh Programmer's Workshop**.

**multiport:** A party-line network structure in which several users share the same line.

**NAU:** See **network addressable unit**.

**network addressable unit (NAU):** LUs, PUs, SSCPs, and the communication links that connect them. NAUs provide services to move information through a network from one user to another and permit the network to be controlled and managed. Each NAU has a network address.

**network ID:** The names by which local LUs, remote LUs, and transaction programs are known throughout the network.

**node:** A processor at the endpoint of any branch of a network.

**node operator:** A function of APPC that permits the control operator to define and control components of a PU 2.1 node.

**NuBus:** A bus specification implemented in the Macintosh II computer. NuBus is a trademark of Texas Instruments.

**parallel sessions:** Multiple LU-LU sessions taking place between the same two LUs concurrently.

**parameter:** A value passed to or from a routine.

**parameter block:** (1) A data structure used to transfer information between applications and certain operating system routines. (2) A set of contiguous memory locations, set up by a calling program to pass parameters to and receive results from an operating system function that it calls.

**partner:** Partner is a MacAPPC term that consists of both a station and control point. It is used by the Configuration program to simplify configuration design and implementation.

**Partner Control Block:** The memory location for the operating parameters of a remote LU that is configured for a specified local LU.

**peer entities:** An entity is an active element within an SNA layer that performs a set of functions in providing its services. Two entities within the same layer are called peer entities.

**peer-to-peer communication:** See **Advanced Program-to-Program Communications**.

**physical unit (PU):** A defined software component of SNA that provides services for managing physical devices on a network and represents those devices to the network. There are various types of PUs, including PU type 2.0 and 2.1, which are used in MacAPPC.

**PIP:** See **program initialization parameters**.

**PIP buffer:** A block of memory allocated and maintained by a transaction program. It is used by MacAPPC device drivers to hold any PIP data that may be sent or received from a partner transaction program.

**pointer:** An item of information consisting of the memory address of some other item.

**polarity:** The contention between two LUs when they both try to allocate a conversation at the same time; for each single or parallel LU-LU session, only one LU is the winner of the session; the other is the loser.



**polling:** The process by which a control station invites tributary stations to send messages. The MacAPPC intelligent communications card is polled during logging and display functions by the Administration program.

**Postmaster:** The part of APPC Mail that sends text and graphics to the Mailbox.

**primary LU (PLU):** The bind sender.

**profile:** A network security setting that may be used for conversation-level checking at the LU or TP levels as a criterion of conversation.

**program initialization parameters (PIP):** Data consisting of one or more subfields, each of which is specified by a separate variable, supplied by the allocating transaction program.

**program-to-program communication:** See **Advanced Program-to-Program Communication**.

**protocol boundary:** A set of routines and parameters, defined as part of SNA, that provides a common language for all application programs and application subsystems.

**PU:** See **physical unit**.

**remote LU:** The LU that is the partner of the local LU of a given transaction program in a conversation. The remote LU may be in the same node as the local LU or in a different node.

**remote resource:** The resources available to transaction programs attached to an LU other than the local LU. Remote is defined in terms of the logical configuration of the network; the LUs can be within the same physical node.

**request buffer:** A block of memory allocated and maintained by the transaction program. It is used by MacAPPC device drivers to communicate with MacAPPC servers.

**request/response unit (RU):** The basic message unit that is transmitted across the SNA network.

**resource:** (1) A part of a network, including physical resources such as printers, keyboards and terminals, and logical resources such as LUs, PUs, modes, sessions, queues, and database records. (2) Data or code stored in a resource file and managed by the Macintosh Resource Manager.

**resource ID:** The name by which a resource is known to the network.

**resource management:** The coordination of the access of transaction programs to various resources, including resources that are local to the LU (such as files, databases, and queues), and shared resources (such as the LU-LU session), to which access is provided through a conversation.

**resynchronization:** Exchange of information concerning resource states between partner LUs to resolve a doubtful situation.

**return code:** System responses that indicate whether a routine request was successful; if an error occurs, the return code indicates the type of error.

**routine:** A formatted function that defines the protocol boundary. In this book, this term is used for MacAPPC functions to distinguish them from APPC verbs.

**RU:** See **request/response unit**.

**SAA:** See **Systems Application Architecture**.

**SDLC:** See **Synchronous Data Link Control**.

**secondary LU (SLU):** The bind receiver.

**security:** The process of allowing or disallowing access to a network via comparison of user IDs, passwords, and profiles.

**server:** A software component that operates on an intelligent processor. MacAPPC protocols are implemented by the MacAPPC server operating on an intelligent communications card residing in a NuBus slot on a Macintosh II computer.

**server window:** The window displayed by the Administration program when you choose Display from the Server menu.

**service transaction program:** A type of SNA-defined program that offers services to application transaction programs, such as translating APPC routines into commands that are understood by the application. DIA is implemented as a service transaction program.

**session:** The resource that enables one LU to communicate with another LU via the SNA path control network. A logical state that exists between two network addressable units and supports a succession of transmissions between them.

**single-session:** The connection of two LUs during one LU-LU session.



**slave:** A computer, terminal, printer, or other peripheral device that responds to being addressed by another computer acting as a master.

**SNA:** See **Systems Network Architecture**.

**SNA Distribution Services (SNADS):** An architecture that allows users on an SNA network to exchange data in an asynchronous fashion, using a "store-and-forward" service.

**SNADS:** See **SNA Distribution Services**.

*SNA Format and Protocol Reference Manual for LU Type 6.2:* IBM's official description of SNA from a design viewpoint; covers LU type 6.2 protocols. Often called the *FAP manual* or *FAPL*.

*SNA Format and Protocol Reference Manual for Type 2.1 Nodes:* IBM's formal description of SNA from a design viewpoint; covers PU type 2.1 protocols.

*SNA Transaction Programmer's Reference Manual for LU Type 6.2:* IBM's formal description of the syntax used to define the APPC protocol boundary from the viewpoint of the transaction program. Often referred to as the *TPRM manual* or *TPRM*.

**SNA/SDLC:** An abbreviation for *Systems Network Architecture/Synchronous Data Link Control*. SNA is a set of rules for controlling the transfer of information in a data communication network. SDLC is a communication-line control protocol that uses commands to control data transfer over a communication line. IBM telecommunications products manufactured after 1978 use this protocol.

**SSCP:** See **system services control point**.

**station:** Hardware and software within an SNA node that control the link connection. This term is synonymous with *link station* and *adjacent link station*. There are two types of stations: primary link stations and secondary link stations.

**subarea nodes:** A node that can communicate with its own peripheral nodes and also other subarea nodes in the network. There are two types of subarea nodes: host (type 5) and controller (type 4).

**subsystem:** A logical "processor" providing an execution environment for transaction programs.

**switched:** A line that is established when required and broken when a session is completed.

**symbol string type A:** A string type consisting of one or more uppercase letters A through Z, numerics 0 through 9, and special characters \$, #, @; the first character is an uppercase letter or a special character.

**symbol string type AE:** A string type consisting of one or more lowercase letters a through z, uppercase letters A through Z, numerics 0 through 9, and special characters the period (.), \$, #, and @; there are no restrictions on the first character.

**synchronous:** (adj.) Unable to continue processing while the original request is still executing. When an application makes a synchronous MacAPPC call, the application does not continue until the MacAPPC routine is complete. Compare **asynchronous**.

**Synchronous Data Link Control (SDLC):** A protocol that specifies the rules that govern the functions performed in the data link control layer of SNA.

**sync-point:** A short term for *synchronization point*. A distributed transaction program error recovery service to "protect" conversations.

**system:** A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a network system comprising a mainframe computer, several microcomputers, printers, modems, communications controller, and several disk drives.

**system services control point (SSCP):** A network addressable unit that provides the services needed to manage an SNA network and establish and control the interconnections needed to permit users to communicate with each other.

**Systems Application Architecture (SAA):** A set of interfaces and protocols developed by IBM and designed to be used by applications that run under different operating systems and in different hardware environments.

**Systems Network Architecture (SNA):** An IBM architecture that describes the logical structure, formats, protocols, and operational sequences for transmitting information units through networks and for controlling the configuration and operation of networks.

**TP:** See **transaction program**.

**TPN:** See **transaction program name**.



**TPRM:** See *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

**transaction:** Two or more processors communicating in the execution of a unit of processing. Each time data is transferred over a network, a transaction occurs.

**transaction program (TP):** A computer program that performs transaction processing. APPC provides services that permit a TP to communicate with other TPs.

**transaction program name (TPN):** A unique name given to a TP for a local LU. The TPN is carried in the allocation request sent by the partner program. The names CNOS and ADMIN are reserved by MacAPPC.

**Transaction Program Control Block (TPCB):** A block of memory allocated and maintained by a transaction program for use by MacAPPC drivers to maintain state and control information about an individual connection to a MacAPPC server.

**transaction program instance:** An execution instance of a transaction program for a particular transaction at a particular LU.

**user ID:** A network security name used to control resource access.

**User Interface Toolbox:** The software in the Macintosh ROM that helps you implement the standard Macintosh user interface in your application.

**verb:** A formatted function that defines the protocol boundary. Transaction programs issue a verb and the LU executes it. In this book, the term *routine* is used for MacAPPC functions to distinguish them from APPC verbs.

**VTAM:** See **Virtual Telecommunications Access Method**.

**Virtual Telecommunications Access Method:** A telecommunications access method that controls the transmission of data to and from local devices that are attached directly by channels, and the communication between a host processor and remote devices via a communications controller. Properly called ACF/VTAM for *Advanced Communications Function for the Virtual Telecommunications Access Method*.

**XCMD:** Abbreviation for external command, a command that extends HyperTalk's command set.

**XData XCMD:** One of the HyperCard APPC external commands that creates the data structures used to pass data between MacAPPC and HyperCard APPC.

**XFNC:** Abbreviation for external function, a function that extends HyperTalk's built-in functions.



# Index

## A

- Access Type 11-6
- activating network components and sessions 12-25 to 12-26
- Adj Station setting, mode configuration 11-38
- Administration program
  - CNOS TP, starting and stopping 12-24
  - configuration file, selecting 12-4
  - control point, display settings for 12-16 to 12-17
  - conventions used in 12-3
  - cursor described 12-3
  - described 2-4, 2-11
  - edit server settings 12-4 to 12-5
  - features of 12-2
  - lines
    - activating 12-25
    - deactivating 12-29
    - display settings for 12-14 to 12-16
  - local LUs
    - activating 12-26
    - deactivating 12-27 to 12-28
    - display settings for 12-7, 12-9 to 12-11
  - logging facility 12-29 to 12-31
  - and MacAPPC Server 2-12 to 2-13
  - menus for 12-2 to 12-3
  - modes
    - activating 12-26
    - deactivating 12-27 to 12-28
    - display settings for 12-8, 12-20 to 12-22
  - network components
    - activating 12-25 to 12-26
    - deactivating 12-26 to 12-29
    - display settings 12-5 to 12-23
  - network display control 12-3
  - remote LUs, display settings for 12-8, 12-18 to 12-19
  - server window
    - display 12-5 to 12-6
    - updating 12-23
  - sessions
    - activating 12-26
    - deactivating 12-27
    - display settings for 12-9, 12-22 to 12-23
  - severity control 12-3, 12-30
  - start a server 12-4 to 12-5
  - stations
    - activating 12-25 to 12-26
    - deactivating 12-29
    - display settings for 12-16 to 12-17
  - stop a server 12-31 to 12-32
  - transaction programs (TPs), display settings for 12-11 to 12-14
- ADMIN reserved TP name 11-27
- ADSP protocol 9-4
- Advanced Program-to-Program Communication. *See* APPC
- allocateState handler (HyperCard APPC) H-25
- allocation errors, result codes listed for C-13 to C-14
- ALS (adjacent link station)
  - ALS Address for partners 11-19, 11-35
  - CNOS ALS and remote LU configuration 11-36
- APDA (Apple Programmer's and Developer's Association) ix to x, H-1
- APPC. *See also* HyperCard APPC; MacAPPC
  - description of 1-2
  - and distributed transaction processing 1-9
  - and LU 6.2 1-5
  - option sets supported by MacAPPC I-1 to I-2
  - PU 2.1 1-6
- APPC.p, listing of A-1 to A-13
- APPC XCMD H-20 to H-21
- appConvState parameter block field 3-8
- APPCErrors.p, listing of B-2 to B-7
- appHiResult
  - parameter block field 3-8
  - values listed for C-2 to C-16
- appLoResult
  - parameter block field 3-8
  - values listed for C-2 to C-16
- appOpCode parameter block field 3-6 to 3-7
- APPCParamBlock data
  - structure 3-5 to 3-8
  - and HyperCard APPC H-24
- appRefNum parameter block field 3-6
- appUserRef parameter block field 3-8
- Apple IPC (Inter-process Communications)
  - component, and the MCP operating system 9-4
- Apple Programmer's and Developer's Association (APDA) ix to x, H-1
- AppleTalk
  - ADSP protocol and 9-4



- and MacAPPC server 2-5 to 2-6
- Apple Technical Library* ix
- ASCII to EBCDIC string conversion 7-10 to 7-11
- table for J-1 to J-2
- AST-ICP communications card 9-2 to 9-3
- asynchronous routine execution 3-4
- attaching
  - defined 2-9
  - routines valid for different attach types 7-3
- attachState handler (HyperCard APPC) H-25

## B

- bad completion errors, result codes listed for C-13
- basic conversations 2-9. *See also* Conversation Driver (.CV62)
- routines listed 3-3
- basic transmission unit length (BTU), configuration of 11-32
- baud rate, for lines 11-32
- binds, queuing of 11-39
- Blank Mode, mode configuration 11-39
- Boolean values, in parameter descriptions 3-12
- bracketing, a data flow control service 1-8
- BTU (basic transmission unit length), configuration of 11-32
- buffering of data, Conversation Driver (.CV62) 4-2

## C

- change-number-of-sessions routines. *See* CNOS routines
- character type conventions, in the Configuration program 11-5
- check boxes, described 11-4
- Chooser desk accessory, select MacAPPC server 10-2 to 1-3
- Clear Log command (Log menu) 12-31
- client computer

- files needed in System Folder of 9-4
- Macintosh computers supported as 9-2
- CNOS ALS setting, remote LU configuration 11-36
- CNOS routines
  - and basic conversations 2-9
  - control operator routines 5-3 to 5-10, 5-49 to 5-50
  - listing of 3-3
  - and MacAPPC Control Operator Driver 2-9
- CNOS TP
  - reserved name 11-27
  - starting and stopping 12-24
- code listings. *See* listings
- communications card
  - illustrated 2-6
  - jumper setting for the AST-ICP 9-3
  - slot number for 12-5
  - specifications of 9-2
- configuration file. *See also* Configuration program
  - printing 11-43
  - sample of 11-44 to 11-45
  - selecting of 12-4
- Configuration program
  - character type conventions 11-5
  - configuration file
    - create 11-5 to 11-6
    - print 11-43
    - sample file 11-44 to 11-45
  - defaults, editing of 11-39 to 11-43
  - delete network components 11-43
  - described 2-11
  - error-checking routine 11-8
  - function described 2-4
  - key conventions 11-4
  - lines
    - creating 11-16 to 11-17
    - editing defaults 11-41
    - editing settings 11-31 to 11-33
  - local LUs
    - creating 11-8 to 11-11
    - editing defaults 11-40
    - editing settings 11-24 to 11-27
  - and MacAPPC Server 2-12 to 2-13

- menus 11-3 to 11-4
- modes
  - creating 11-22 to 11-23
  - editing defaults 11-43
  - editing settings 11-37 to 11-39
- network components
  - creating 11-7 to 11-24
  - delete 11-43
  - described 11-2
  - editing settings 11-24 to 11-39
  - name types 11-5
- node setting defaults, editing 11-40
- overview of 11-2
- partners
  - creating 11-18 to 11-19
  - editing defaults 11-42
  - editing settings 11-33 to 11-35
- print a configuration file 11-43
- remote LUs
  - creating 11-20 to 11-21
  - editing defaults 11-42
  - editing settings 11-35 to 11-37
- screen conventions 11-4
- transaction programs
  - creating 11-12 to 11-16
  - editing defaults 11-41
  - editing settings 11-27 to 11-31
- worksheets for K-1 to K-3
- connection routines, listing of 3-3
- connections types, MacAPPC support of 2-7 to 2-8
- Connect Type 11-32
- constants
  - for completion of routine 3-8
  - for Control Operator Driver (.CO62) 5-45 to 5-46
  - for Conversation Driver (.CV62) 4-63 to 4-64
  - for MacAPPC drivers 3-10
  - for Node Operator Driver (.NO62) 6-35 to 6-36
  - for the Transaction Program Driver (.TP62) 7-12
- control blocks 3-8 to 3-9
- Control Operator Driver (.CO62)
  - CNOS routines
    - described 5-3 to 5-10



- summarized 5-49 to 5-50
- constants available 5-45 to 5-46
- data types 5-47 to 5-48
- LU definition routines
  - described 5-14 to 5-44
  - summarized 5-52 to 5-56
- parameter block for 3-4
- parameter mapping F-2 to F-8
- result codes listed for C-15
- routines
  - categories of 2-9, 5-3
  - functions described 2-4
  - listed 3-3
  - and transaction program
    - configuration 11-29
  - security and 2-7, 5-2 to 5-3
  - session control routines
    - described 5-11 to 5-13
    - summarized 5-51
- control point. *See* CP Name
- Conversation Control Block (CVCB) 3-9
- Conversation Driver (.CV62)
  - basic conversation routines
    - description of 4-38 to 4-62
    - states for 4-4
    - summarized 4-73 to 4-77
  - buffering of data 4-2
  - constants available for 4-63 to 4-64
  - conversation routine types
    - defined 2-8 to 2-9
  - conversation states defined 4-3
  - conversation types 4-2
  - data formatting 4-2
  - data mapping 4-5
  - data types 4-65
  - mapped conversation routines
    - description of 4-8 to 4-32
    - states for 4-3
    - summarized 4-67 to 4-71
  - mapping parameter block 4-5 to 4-6, 4-66
  - parameter block for 3-4
  - parameter mapping E-2 to E-11
  - routines
    - functions described 2-4
    - listed 3-3
    - routine mapping D-2 to D-3
  - type-independent
    - conversation routines

- description of 4-33 to 4-37
- states for 4-4
- summarized 4-72
- writing a mapping procedure 4-5 to 4-7
- conversation handlers
  - (HyperCard APPC) H-26 to H-27
- conversation ID 3-11
  - returned by MCAAllocate routine 4-9
- conversation-level security 2-7, 5-2 to 5-3. *See also* security
- LU Security 11-26
- transaction program
  - configuration 11-14, 11-29
- conversation routine mapping D-2 to D-3
- conversations. *See also* Conversation Driver (.CV62)
- failure of and bracketing 1-8
- and MacAPPC Conversation Driver 2-8 to 2-9
- and transaction program (TP) 1-7 to 1-8
- Conv Type (conversation type), transaction program
  - configuration 11-12, 11-28
- .CO62 driver. *See* Control Operator Driver (.CO62)
- CP Name setting
  - display settings for station and control point 12-16 to 12-18
  - remote LU configuration 11-36
- CPU ID setting, for partner 11-18 to 11-19, 11-34
- Create menu
  - create network components 11-8
  - for Configuration program 11-3
- cursor, display of in
  - Administration program 12-3
- .CV62 driver. *See* Conversation Driver (.CV62)

## D

- data flow control service, and bracketing 1-8
- data formatting, mapped vs. basic conversations 2-9

- Data Mapping, transaction
  - program configuration 11-29
- data streams
  - and basic conversation routines 4-38
  - defined 1-6
- data types
  - for Control Operator Driver (.CO62) 5-47 to 5-48
  - for Conversation Driver (.CV62) 4-65
  - for HyperCard APPC H-23 to H-24
  - MacAPPC data types and the xDefine command H-40
  - for Node Operator Driver (.NO62) 6-37 to 6-38
  - for Transaction Program Driver (.TP62) 7-13
- deactivating network
  - components and sessions 12-26 to 12-29
- deallocation errors, result codes listed for C-15
- defineCOandNO handler (HyperCard APPC) H-24
- defineCVandTP handler (HyperCard APPC) H-23 to H-24
- delete, network components 11-43
- Device Manager, and parameter block fields 3-5
- DIA (Document Interchange Architecture), APPC and 1-3
- DISOSS (Distributed Office Support System) 1-3
- Display command (Server menu) 12-5
- Distributed Office Support System (DISOSS) 1-3
- distributed transaction
  - processing
    - description of 1-9
    - and SNA network 1-2
- Document Interchange Architecture (DIA), APPC and 1-3
- drivers. *See* MacAPPC drivers
- Duplex Type setting 11-33

## E

- EBCDIC to ASCII string
  - conversion 7-10 to 7-11
  - table for J-1 to J-2



- Edit menu
  - for Administration program 11-2
  - for Configuration program 11-3
- Enter key, use in Configuration program described 11-4
- error result codes. *See* result codes
- errors file, for MacAPPC drivers B-2 to B-7
- errStr XCMD H-22
- Exchange ID setting
  - for local node 11-6
  - for partner 11-18, 11-34
- exec handlers (HyperCard APPC) H-25

## F

- FALSE, defined 3-12
- File menu
  - for Administration program 11-2
  - for Configuration program 11-3
- New command 11-5
- files, provided on disks 9-3 to 9-4
- Finder version requirement 9-3
- FMH Data setting, transaction program configuration 11-29
- Forwarder (FWD), and the MCP operating system 9-4
- full duplex 11-33
- FWD (Forwarder), and the MCP operating system 9-4

## G

- General Data Stream (GDS) variables, and mapped conversations 2-9
- get62Srvr XCMD H-21 to H-22

## H

- half duplex 11-33
- handlers
  - for HyperCard APPC H-23 to H-27
  - sample use of H-37 to H-39
- hardware installation 9-2 to 9-3

- HyperCard APPC. *See also* MacAPPC; XCMDs and XFCNs
- APPC Mail sample application H-11 to H-14
- application development H-20 to H-27
- Application Programming Interface (API) H-3 to H-4
- Conversation ID field H-9
- data types H-23 to H-24
- error result codes H-8
- handlers H-23 to H-27
- hardware requirements H-2
- help for the Lab H-9 to H-11
- Lookup Chooser Server and Zone button H-9
- MacAPPC data types and the xDefine command H-40 to H-41
- MacAPPC Lab H-4 to H-11
- sample session H-15 to H-19
- Mailbox H-14
- navigation button H-4
- navigation cards for the Lab H-5
- overview of H-3 to H-14
- parameters for XData XCMDs and XFCNs H-36 to H-37
- Postmaster H-13 to H-14
- Program ID field H-9
- Receive data field H-9
- record types H-24
- returned values parameters H-7 to H-8
- routine cards H-6
- scripts H-23 to H-27
- software requirements H-2
- supplied values parameters H-7
- title card H-3
- TPAttach card H-6, H-7
- XData errors H-39
- XData XCMDs and XFCNs H-28 to H-36

## I

- IBM Corporation
  - documents about APPC and LU 6.2 x
  - history of SNA 1-2
- IBM equipment
  - local node exchange ID for 11-6
  - partner exchange ID 11-18

- IBM 3270 data stream 4-38
- IBM 5250 data stream 4-38
- Idle Time setting 11-33
- IDs, for MacAPPC drivers 3-11
- I-frames setting, maximum number of 11-33
- Init Q Req setting, remote LU configuration 11-36
- installation
  - hardware 9-2 to 9-3
  - software 9-3 to 9-5
- interface file for MacAPPC drivers A-2 to A-13
- International Business Machines. *See* IBM Corporation
- interprogram communication, and LU 6.2 1-5
- io parameter block fields 3-5 to 3-6

## J

- jumper settings for the AST-ICP communications card 9-3

## K

- k (lowercase), to specify routine name for appcOpCode parameter block field 3-6 to 3-7
- key conventions, in the Configuration program 11-4

## L

- Lcl Sec (local security) setting, remote LU configuration 11-37
- leased connection type configuration of 11-32
- MacAPPC support of 2-7 to 2-8
- Line ... menu option, described 11-2
- Line Name setting 11-32
- Line Number setting 11-32
- lines
  - activating 12-25
  - configuration of 11-16 to 11-17
- Connect Type setting 11-32
- deactivating 12-29



- display settings for 12-14 to 12-16
- Duplex Type setting 11-33
- editing defaults 11-41
- editing settings 11-31 to 11-33
- Idle Time setting 11-33
- Line Name setting 11-32
- Line Number setting 11-32
- Line Speed setting 11-32
- Line Type setting 11-32
- LOCAL name reserved 11-7, 11-17, 11-32, 11-34
- Max BTU setting 11-32
- Max I-Frames setting 11-33
- Max Retries setting 11-32
- NP Recv Time setting 11-33
- NRZI Support setting 11-33
- Role Type setting 11-32
- Line Speed setting 11-32
- Line Type setting 11-32
- link connections, MacAPPC support of 2-7
- listings
  - default mapping procedure 4-7
  - errors file B-2 to B-7
  - examples in a transaction program 8-2 to 8-10
  - get currently chosen MacAPPC server 7-2 to 7-3
  - interface file for MacAPPC drivers A-2 to A-13
  - LOCAL line name reserved 11-7, 11-17, 11-32, 11-34
  - Local LU ... menu option, described 11-2
- local LUs
  - activating 12-26
  - creating 11-8 to 11-11
  - deactivating 12-27 to 12-28
  - display settings for 12-7, 12-9 to 12-11
  - editing defaults 11-40
  - editing settings 11-24 to 11-27
  - Local LU name setting 11-25
  - LU ID setting 11-8, 11-25
  - LU Security setting 11-26
  - maximum number of TPs 11-9
  - Max Sessions setting 11-8, 11-25
  - Name setting 11-8
  - Net Name setting 11-25

- Net Qual setting 11-25
- Password setting 11-9, 11-27
- Profiles setting 11-10, 11-26
- transaction programs for 11-12 to 11-16
- User ID setting 11-9, 11-26
- Wait Time setting 11-26
- Local Node, create new configuration file 11-6
- local resources 1-7
- logging facility 12-29 to 12-31
- settings options 12-29 to 12-30
- logical unit of work (LUW) IDs, transaction program configuration 11-29
- logical units (LUs) 1-3. *See also* local LUs; remote LUs and sessions 1-7
- Log menu
  - Clear Log command 12-31
  - commands for 12-29
  - for Administration program 12-3
  - Settings command 12-30
  - Show Log command 12-31
- log window 12-31
- LU definition routines
  - listing of 3-3
  - and the MacAPPC Control Operator Driver 2-9
- LU ID setting, for local LUs 11-8, 11-25
- LU Security setting, for local LUs 11-26
- LU 6.2
  - control operator parameter mapping F-2F.8
  - conversation parameter mapping E-2 to E-11
  - description of 1-5 to 1-6
  - and primary session unit 1-3
  - protocol boundary 1-6
  - resource allocation 1-7
  - return code mapping G-2 to G-3
  - routine mapping D-1 to D-4
- LUW (logical unit of work) setting, transaction program configuration 11-29

## M

- MacAPPC. *See also* APPC; HyperCard APPC

- APPC option sets supported by I-1 to I-2
- applications provided 2-4
- connection types 2-8
- connectivity and 2-3 to 2-4
- drivers described 2-8 to 2-10
- environment illustrated 2-3
- link connections supported by 2-7
- and Macintosh user interface 2-5
- network structure, illustrated 2-2
- purpose of 2-2
- routine categories 2-4
- security levels 2-7
- server-client architecture and 2-5 to 2-6
- software and server relationship 2-11 to 2-13
- and transaction programs (TPs) 2-4
- transmission media 2-7 to 2-8
- utilities provided 2-4
- MacAPPC Chooser device
  - function described 2-4
- MacAPPC server as an entity name in resource fork 7-2
- MacAPPC drivers 2-8 to 2-10. *See also individual driver names*
  - constants listed 3-10
  - control blocks 3-8 to 3-39
  - errors file (Pascal) B-2 to B-7
  - executing a driver routine 3-11
  - IDs for 3-11
  - interface file for (Pascal) A-2 to A-13
  - listed 3-2
  - opening of 3-2
  - parameter blocks for 3-4 to 3-8
  - synchronous and asynchronous program execution 3-4
  - typographic conventions used 3-12
- MacAPPC routines, specify execution of 3-6 to 3-7
- MacAPPC server
  - and the AppleTalk network 2-5 to 2-6
  - card illustrated 2-6
  - function described 2-4



- get currently selected
  - MacAPPC server (procedure for) 7-2 to 7-3
- select using the Chooser 10-2 to 1-3
- software and server relationship 2-11 to 2-13
- starting 12-4 to 12-5
- stopping 12-31 to 12-32
- MacAPPC System disk, contents of 9-4
- MacAPPC User disk, contents of 9-3
- Macintosh computers
  - documents about ix to x
  - types supported as client computers 9-2
  - types supported as server computers 9-2
  - types supported for HyperCard APPC H-2
- Macintosh Coprocessor Platform operating system 9-3, 9-4
- Macintosh user interface, and MacAPPC 2-5
- management of the network.
  - See Administration program manual
  - overview of vii
  - structure of viii
  - typographic conventions used xi
- mapped conversation buffer control block 3-9
- mapped conversations 2-9.
  - See also Conversation Driver (.CV62)
  - examples in a transaction program 8-2 to 8-10
  - mapped routines listed 3-3
- mapper (user-supplied mapping utility) 4-5
- mapping parameter block 4-5 to 4-6, 4-66
- mappings
  - control operator parameter mapping F-2F.8
  - conversation parameter mapping E-2 to E-11
  - result codes G-2 to G-3
  - routine mapping D-1 to D-4
- Max BTU setting 11-32
- Max I-Frames setting 11-33
- Max RU LB setting, mode configuration 11-38

- Max RU UB setting, mode configuration 11-38
- Max Sess (maximum number of sessions) setting
  - for local LUs 11-8, 11-25
  - mode configuration 11-22, 11-39
- Max TPs (maximum number of TPs) setting, for local LUs 11-9, 11-26
- memory requirement, for HyperCard APPC H-2
- Memory Size, edit server settings 12-4 to 12-5
- menus
  - for the Administration program 12-2 to 12-3
  - for the Configuration program 11-3 to 11-4
- message units, request/response units (RUs) 1-3
- Min 1st Spkrs (minimum number of first speakers) setting, for modes 11-22, 11-39
- Mode ... menu option, described 11-2
- Mode Name setting 11-37
- modes
  - activating 12-26
  - Adj Station setting 11-38
  - Blank Mode setting 11-39
  - creating 11-22 to 11-23
  - deactivating 12-27 to 12-28
  - display settings for 12-8, 12-20 to 12-22
  - editing defaults 11-43
  - editing settings 11-37 to 11-39
  - Max RU LB setting 11-38
  - Max RU UB setting 11-38
  - Max Sessions setting 11-22, 11-39
  - Min 1st Spkrs setting 11-22, 11-39
  - Mode Name setting 11-37
  - name for 11-22
  - PB Sessions setting 11-23, 11-39
  - Queue Binds setting 11-39
  - Recv Pacing setting 11-38
  - Send Pacing setting 11-38
  - Session Reinit setting 11-38

- Sync Level setting 11-22, 11-38
- Monitor Timer setting 11-6
- multipoint connection type configuration of 11-32
- MacAPPC support of 2-7 to 2-8

## N

- name
  - for lines 11-16 to 11-17
  - for local LUs 11-8
  - for mode 11-22
  - for partner 11-18
  - for remote LU 11-20
  - for transaction program 11-12
- NAUs. See network addressable units (NAUs)
- negotiable SDLC role 11-32
- Net Name setting
  - local LU configuration 11-25
  - remote LU configuration 11-36
  - transaction program configuration 11-27
- Net Qual setting
  - local LU configuration 11-25
  - remote LU configuration 11-36
- network addressable units (NAUs)
  - components of 1-3
  - NAU services and the SNA network 1-5
- network display control, in the Administration program 12-3
- network management. See Administration program
- New command (File menu) 11-5
- NIL pointer, defined 3-12
- Node ... menu option, described 11-2
- Node Operator Driver (.NO62)
  - constants available for 6-35 to 6-36
  - data types for 6-37 to 6-38
- node control routines
  - described 6-2 to 6-10
  - summarized 6-39 to 6-40
- node definition routines
  - described 6-17 to 6-34
  - summarized 6-42 to 6-44
- node message routines
  - described 6-11 to 6-16



- summarized 6-41
- parameter block for 3-4
- result codes listed for C-16
- routines
  - categories of 2-9, 6-2
  - functions described 2-4
  - listed 3-3
- nodes
  - editing defaults 11-40
  - and the SNA network 1-3
- nonproductive receive time 11-33
- nonreturn-on-zero encoding methods 11-33
- .NO62 driver. *See* Node Operator Driver (.NO62)
- NP Recv Time setting 11-33
- NRZI Support setting 11-33
- NuBus card. *See* communications card
- null mode name, send across a link 11-39
- NULL value, defined 3-12

## O

- operating system, MCP 9-3, 9-4
- option sets supported by MacAPPC I-1 to I-2

## P

- pacing response, and mode configuration 11-38
- parallel sessions 1-7
  - and remote LU configuration 11-20, 11-36
- parameter blocks
  - for MacAPPC drivers 3-4 to 3-8
  - mapping parameter block 4-5 to 4-6
- Partner ... menu option, described 11-2
- Partner Name setting 11-34
- Partner Node, create new configuration file 11-7
- partners
  - ALS address setting 11-19, 11-35
  - CPU ID setting 11-18 to 11-19
  - creating 11-18 to 11-19
  - editing defaults 11-42

- editing settings 11-33 to 11-35
- Exchange ID setting 11-18, 11-34
- Line Name setting 11-34
  - name for 11-18
- Partner Name setting 11-34
- Phone Number setting 11-35
- Pascal procedures. *See* listings
- password setting
  - and conversation level access security 2-7
  - and local LU configuration 11-9, 11-27
  - and remote LU configuration 11-36
- path-control network services, and the SNA network 1-5
- PB Sessions (number of prebound sessions) setting, mode configuration 11-23, 11-39
- Phone Number setting, partner configuration 11-35
- physical units (PUs) 1-3. *See also* PU 2.1
- PIP buffer control block 3-9
- PIP Check setting, and transaction program configuration 11-28
- PIP Count setting, and transaction program configuration 11-28
- PIP (program initialization parameters) setting, and transaction program configuration 11-28
- polarity, and LU contention 1-7
- prebound sessions for modes 11-23, 11-39
- primary SDLC role 11-32
- primary session unit 1-3
- print, a configuration file 11-43
- Privilege setting, and transaction program configuration 11-29
- procedures. *See* listings
- profile setting
  - and conversation level access security 2-7, 5-2
  - and local LU configuration 11-10, 11-26
  - and transaction program configuration 11-14 to 11-15, 11-30 to 11-31

- Prof security setting, and transaction program configuration 11-14, 11-29
- program errors. *See also* result codes
  - result codes listed for C-14
- program ID 3-11
- program-to-program communication 1-5
- protocol boundary, LU 6.2 1-6
- PU 2.1, and APPC 1-6

## Q

- qLink parameter block field 3-5
- qType parameter block field 3-5
- Queue Binds setting, mode configuration 11-39
- queue session-initiation requests 11-36

## R

- radio buttons, described 11-4
- Recv Pacing setting, mode configuration 11-38
- reference materials ix to x
  - for HyperCard H-1
- Remote LU ... menu option, described 11-2
- remote LUs
  - CNOS ALS setting 11-36
  - CP Name setting 11-36
  - creating 11-20 to 11-21
  - display settings for 12-8, 12-18 to 12-19
  - editing defaults 11-42
  - editing settings 11-35 to 11-37
  - Init Q Req setting 11-36
  - Lcl Sec (local security) setting 11-37
  - name for 11-20
  - Net Name setting 11-36
  - Net Qual setting 11-36
  - Parallel Sessions setting 11-20, 11-36
  - Password setting 11-36
  - Remote LU setting 11-35
- remote resources 1-7
- request/response units (RUs) 1-3
  - and mode configuration 11-38



- reserved names
  - ADMIN TP name 11-27
  - CNOS TP name 11-27
  - LOCAL line name 11-7, 11-17, 11-32, 11-34
  - SNASVCMG mode name 11-37
- resource-access-level security.
  - See also* security
  - and transaction program configuration 11-14, 11-29
- resource allocation, and LU 6.2 1-7
- result codes
  - for asynchronous routine execution 3-4
  - for control operator errors C-15
  - for HyperCard APPC H-8
  - mapping of G-2 to G-3
  - for node operator errors C-16
  - for program errors C-14
  - for state errors C-13
  - for usage errors C-2 to C-13
  - values listed for C-2 to C-16
- retries, maximum number of 11-32
- return codes (LU 6.2),
  - MacAPPC result code mapping G-2 to G-3
- Return key, use in
  - Configuration program described 11-4
- Role Type setting 11-32
- routine handlers (HyperCard APPC) H-25
- routine mapping D-1 to D-4
- routines. *See under individual drivers*
- RUs. *See* request/response units (RUs)

## S

- screen conventions, in the
  - Configuration program 11-4
- scripts, for HyperCard APPC H-23 to H-27
- SDLC. *See* Synchronous Data Link Control (SDLC)
- secondary SDLC role 11-32
- secondary session unit 1-3
- security
  - levels provided 2-7, 5-2 to 5-3
- and remote LU configuration 11-37
- for transaction programs 11-13 to 11-14, 11-29 to 11-30
- Send Pacing setting, mode configuration 11-38
- serial port number, for line connection 11-32
- server-client architecture, and MacAPPC 2-5 to 2-6
- server computer. *See also* MacAPPC server
  - files needed in System Folder of 9-4
  - Macintosh II supported as 9-2
- Server menu
  - for Administration program 12-3
  - Display command 12-5
  - Start CNOS command 12-24
  - Start Server command 12-4
  - Update command 12-23
- Server Name, edit server settings 12-4 to 12-5
- server window 12-6
- updating 12-23
- session-control routines, and the MacAPPC Control Operator Driver 2-9
- session control routines, listing of 3-3
- session ID 3-11
- session level LU-LU verification 2-7, 5-2 to 5-3. *See also* security
- Session Reinit setting, mode configuration 11-38
- sessions
  - activating 12-26
  - and conversation failure 1-8
  - deactivating 12-27
  - defined 1-3
  - display settings for 12-9, 12-22 to 12-23
  - maximum number of sessions for local LUs 11-8, 11-25
  - maximum number of sessions for modes 11-22, 11-39
  - parallel sessions and remote LU configuration 11-20, 11-36
  - prebound sessions for modes 11-23

- Settings command (Log menu) 12-30
- setupCVData handler (HyperCard APPC) H-25
- setupTPData handler (HyperCard APPC) H-25
- severity control
  - in the Administration program 12-3
  - settings for 12-30
- Show Log command (Log menu) 12-31
- single session 1-7
- reinitiation and mode configuration 11-38
- Slot, edit server settings 12-4 to 12-5
- SNA
  - components of 1-3 to 1-4
  - and end user 1-3
  - functional layers 1-5
  - IBM's development of 1-2
  - logical components of 1-3 to 1-4
  - physical components of 1-3 to 1-4
- SNA data streams 4-38
- SNA Distribution Services (SNADS), APPC and 1-3
- SNADS (SNA Distribution Services) 1-3
- SNASVCMG mode
  - activating 12-26
  - CNOS and 12-24
  - deactivating 12-27
  - reserved name 11-37
- software installation 9-3 to 9-5
- speakers, minimum number of
  - first speakers for modes 11-22, 11-39
- SSCP (system services control points) 1-3
- Start CNOS command (Server menu) 12-24
- Start Server command (Server menu) 12-4
- state errors, result codes listed for C-13
- station display window 12-17
- Station Name 12-17. *See also* CP Name
- stations
  - activating 12-25 to 12-26
  - deactivating 12-29



- Status, and transaction program configuration 11-28
- stop
  - CNOS 12-24
  - deactivate network
    - components and sessions 12-26 to 12-29
  - MacAPPC server 12-31 to 12-32
- switched connection type
  - configuration of 11-32
  - MacAPPC support of 2-7 to 2-8
- symbol-string character types 11-5
- Synchronous Data Link Control (SDLC)
  - and APPC 2-7
  - Local Node configuration 11-6
  - and Role Type 11-32
- synchronous routine execution 3-4
- Sync Level (synchronization level) setting
  - mode configuration 11-22, 11-38
  - and transaction program configuration 11-12, 11-28
- System Folder, files needed in 9-4
- system services control points (SSCP) 1-3
- Systems Network Architecture. *See* SNA
- System version 6.0.2
  - requirement 9-3

## T

- Tab key, use in Configuration program described 11-4
- technical reference materials ix to x
  - about HyperCard H-1
- Token Ring, and APPC 2-7
- TP ... menu option, described 11-2
- TP Name setting 11-27
- .TP62 driver. *See* Transaction Program Driver (.TP62)
- Transaction Program Control Block (TPCB) 3-8 to 3-9
- Transaction Program Driver (.TP62)
  - connection routines

- described 7-4 to 7-8
- summarized 7-14
- constants for 7-12
- data types 7-13
- parameter block for 3-4
- routines
  - categories of 2-9 to 2-10, 7-4
  - functions described 2-4
  - listed 3-3
  - routine mapping D-4
- utility routines
  - described 7-9 to 7-12
  - summarized 7-15
- transaction programs (TPs). *See also* Transaction Program Driver (.TP62)
  - APPC and 1-7, 1-8
  - conversations 1-7 to 1-8
  - Conv Type setting 11-12, 11-28
  - creating for local LU 11-12 to 11-16
  - Data Mapping setting 11-29
  - display settings for 12-11 to 12-14
  - editing defaults 11-41
  - editing settings 11-27 to 11-31
  - example fragments of a TP 8-2 to 8-10
  - FMH Data setting 11-29
  - LUW setting 11-29
  - and MacAPPC 2-4
  - and the MacAPPC Transaction Program Driver 2-9 to 2-10
  - maximum number of for local LUs 11-9
  - Name setting 11-12
  - Net Name setting 11-27
  - PIP setting 11-28
  - PIP Check setting 11-28
  - PIP Count setting 11-28
  - Privilege setting 11-29
  - Profile setting 11-14 to 11-15, 11-30 to 11-31
  - security level 11-13 to 11-14, 11-29 to 11-30
  - sessions and 1-7
  - and SNA logical components 1-3 to 1-4
  - Status setting 11-28
  - Sync Level setting 11-12, 11-28
  - TP Name setting 11-27

- User ID setting 11-14, 11-30
- TRUE, defined 3-12
- Type AE symbol-string type 11-5
- Type A symbol-string type 11-5
- Type hexadecimal
  - symbol-string type 11-5
- type-independent
  - conversations 2-9. *See also* Conversation Driver (.CV62)
  - routines listed 3-3

## U

- Update command (Server menu) 12-23
- usage errors, result codes listed for C-2 to C-13
- User ID setting
  - and conversation level access security 2-7
  - local LU configuration 11-9, 11-26
  - and the profile 11-10
  - transaction program configuration 11-14 to 11-15, 11-30
- User/Prof security setting. *See also* security
  - and transaction program configuration 11-14, 11-30
- User security setting. *See also* security
  - and transaction program configuration 11-14, 11-29
- utility routines, listing of 3-3

## V

- VTAM host, partner CPU ID 11-19

## W

- Wait Time setting, for local LUs 11-26
- wake-up interval, for program monitor 11-6
- worksheets for configuration K-1 to K-3

## X

- XCMDs and XFCNs
  - APPC XCMD H-20 to H-21



errStr XCMD H-22  
get62Srvr XCMD H-21 to  
H-22  
Ptr H-32  
xConst XFCN H-22  
xDefine H-29  
xDispose H-33  
xFill H-33  
xGet H-32  
xGlobal H-30  
xLock H-32  
xMode H-34  
xPut H-31  
xResource H-34 to H-35  
xSize H-33  
XData XCMDs and XFCNs  
H-28 to H-36  
*fieldSpec* parameter H-36  
handler sample H-37 to H-39  
MacAPPC data types and  
xDefine command H-40  
to H-41  
*priType* parameter H-37



# Index of Routines, Parameters, and Constants

## **b**

BAllocate routine 4-39 to 4-42  
    summarized 4-73  
BCConfirm routine 4-43  
    summarized 4-73  
BCConfirmed routine 4-44  
    summarized 4-73 to 4-74  
BCDeallocate routine 4-45 to 4-46  
    summarized 4-74  
BCFlush routine 4-47  
    summarized 4-74  
BCGetAttributes routine 4-48 to 4-50  
    summarized 4-74 to 4-75  
BCPostOnReceipt routine 4-51  
    summarized 4-75  
BCPrepareToReceive routine 4-52  
    summarized 4-75  
BCReceiveAndWait routine 4-53 to 4-54  
    summarized 4-75  
BCReceiveImmediate routine 4-55 to 4-56  
    summarized 4-76  
BCRequestToSend routine 4-57  
    summarized 4-76  
BCSendData routine 4-58 to 4-59  
    summarized 4-76  
BCSendError routine 4-60 to 4-61  
    summarized 4-76 to 4-77  
BCTest routine 4-62  
    summarized 4-77

## **coA**

coActBdrs parameter  
    in CODisplayMode 5-35

coActFirstSpkrs parameter  
    in CODisplayMode 5-35  
COActivateSession routine 5-12  
    summarized 5-51  
coActLUSess parameter  
    in CODisplayLocalLU 5-32  
coActSess parameter  
    in CODisplayMode 5-34  
coALSName parameter  
    in CODEfineMode 5-19  
    in CODisplayMode 5-34

## **coB**

coBlankMode parameter  
    in CODEfineMode 5-21  
    in CODisplayMode 5-35

## **coC**

COChangeSessionLimit routine 5-4 to 5-5  
    summarized 5-49  
coCNOSALSName parameter  
    in CODEfineRemoteLU 5-23  
    in CODisplayRemoteLU 5-38  
coConvID parameter  
    in CODisplaySession 5-39  
coConvSecType parameter  
    in CODEfineLocalLU 5-17  
    in CODisplayLocalLU 5-32  
coConvType parameter  
    in CODEfineTP 5-29  
    in CODisplayTP 5-44  
coCPName parameter  
    in CODEfineRemoteLU 5-23  
    in CODisplayRemoteLU 5-38  
coCurMaxSess parameter  
    in COChangeSessionLimit 5-4  
    in CODisplayMode 5-34  
    in COInitializeSessionLimit 5-6

coCurMinBdrs parameter  
    in COChangeSessionLimit 5-5  
    in CODisplayMode 5-34  
    in COInitializeSessionLimit 5-7  
coCurMinFirstSpkrs parameter  
    in COChangeSessionLimit 5-4  
    in CODisplayMode 5-34  
    in COInitializeSessionLimit 5-7

## **coD**

coDataMapping parameter  
    in CODEfineTP 5-29  
    in CODisplayTP 5-44  
CODEactivateSession routine 5-13  
    summarized 5-51  
coDeactType parameter  
    in CODEactivateSession 5-13  
CODEfineLocalLU routine 5-15 to 5-17  
    summarized 5-52  
CODEfineMode routine 5-18 to 5-21  
    summarized 5-52  
CODEfineRemoteLU routine 5-22 to 5-24  
    summarized 5-53  
CODEfineTP routine 5-25 to 5-29  
    summarized 5-53  
coDefLUMaxSes parameter  
    in CODEfineLocalLU 5-17  
    in CODisplayLocalLU 5-32  
coDefMaxSess parameter  
    in CODEfineMode 5-19  
    in CODisplayMode 5-34  
coDefMinBdrs parameter  
    in CODisplayMode 5-34  
coDefMinFirstSpkrs parameter  
    in CODEfineMode 5-19



- in CODisplayMode 5-34
- coDefPBFirstSpkrs parameter
  - in CODEfineMode 5-20
  - in CODisplayMode 5-34
- CODelete routine 5-30
  - summarized 5-54
- CODisplayLocalLU routine 5-31 to 5-32
  - summarized 5-54
- CODisplayMode routine 5-33 to 5-36
  - summarized 5-54 to 5-55
- CODisplayRemoteLU routine 5-37 to 5-38
  - summarized 5-55
- CODisplaySession routine 5-39 to 5-40
  - summarized 5-55 to 5-56
- CODisplayTP routine 5-41 to 5-44
  - summarized 5-56
- coDrainLclLU parameter
  - in CODisplayMode 5-36
- coDrainRmtLU parameter
  - in CODisplayMode 5-36
- coDrainSrc parameter
  - in COResetSessionLimit 5-9
- coDrainTgt parameter
  - in COResetSessionLimit 5-10

## **coE**

- coEnableType parameter
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43

## **coF**

- coFMHDataSupp parameter
  - in CODEfineTP 5-27
  - in CODisplayTP 5-43
- coForceRst parameter
  - in COResetSessionLimit 5-10

## **col**

- COInitializeSessionLimit routine 5-6
  - summarized 5-49

## **col**

- coLclLUID parameter
  - in CODEfineLocalLU 5-17
  - in CODisplayLocalLU 5-32
- coLclLUName parameter

- in CODEfineLocalLU 5-16
- in CODEfineMode 5-19
- in CODEfineRemoteLU 5-23
- in CODEfineTP 5-26
- in CODElete 5-30
- in CODisplayLocalLU 5-31
- in CODisplayMode 5-33
- in CODisplayRemoteLU 5-37
- in CODisplaySession 5-39
- in CODisplayTP 5-41
- coLclProgName parameter
  - in CODEfineTP 5-26
  - in CODElete 5-30
  - in CODisplayTP 5-42
- coLclSecAcc parameter
  - in CODEfineRemoteLU 5-24
  - in CODisplayRemoteLU 5-38
- coLUActive
  - in CODisplayLocalLU 5-32
- coLUPswdOp parameter
  - in CODEfineRemoteLU 5-24
- coLUPswd parameter
  - in CODEfineRemoteLU 5-24
  - in CODisplayRemoteLU 5-38
- coLUWSupp parameter
  - in CODEfineTP 5-27
  - in CODisplayTP 5-43

## **coM**

- coMaxRUHiBound parameter
  - in CODEfineMode 5-20
  - in CODisplayMode 5-34
- coMaxRULoBound parameter
  - in CODEfineMode 5-20
  - in CODisplayMode 5-34
- coMaxTP parameter
  - in CODEfineLocalLU 5-17
  - in CODisplayLocalLU 5-32
- coModeName parameter
  - in COActivateSession 5-12
  - in COChangeSessionLimit 5-4
  - in CODEactivateSession 5-13
  - in CODEfineMode 5-19
  - in CODElete 5-30
  - in CODisplayMode 5-34
  - in CODisplaySession 5-39
  - in COInitializeSessionLimit 5-6
  - in COProcessSessionLimit 5-8
  - in COResetSessionLimit 5-9

## **coN**

- coNetNameOp parameter
  - in CODEfineLocalLU 5-16

- in CODEfineRemoteLU 5-23
- in CODEfineTP 5-26
- coNetName parameter
  - in CODEfineLocalLU 5-16
  - in CODEfineRemoteLU 5-23
  - in CODEfineTP 5-26
  - in CODisplayLocalLU 5-31
  - in CODisplayRemoteLU 5-37
  - in CODisplayTP 5-42
- coNetQualOp parameter
  - in CODEfineLocalLU 5-16
  - in CODEfineRemoteLU 5-23
- coNetQual parameter
  - in CODEfineLocalLU 5-16
  - in CODEfineRemoteLU 5-23
  - in CODisplayLocalLU 5-32
  - in CODisplayRemoteLU 5-38
- coNextLclLUName parameter
  - in CODisplayLocalLU 5-31
- coNextLclProgName parameter
  - in CODisplayTP 5-41
- coNextModeName parameter
  - in CODisplayMode 5-34
- coNextRmtLUName parameter
  - in CODisplayRemoteLU 5-37
- coNextSessID parameter
  - in CODisplaySession 5-39
- coNextUserName parameter
  - in CODisplayLocalLU 5-32
  - in CODisplayTP 5-42

## **coP**

- coParSess parameter
  - in CODEfineRemoteLU 5-24
  - in CODisplayRemoteLU 5-38
- coPIPCheck parameter
  - in CODEfineTP 5-27
  - in CODisplayTP 5-42
- coPIPCount parameter
  - in CODEfineTP 5-26
  - in CODisplayTP 5-42
- coPIPReq parameter
  - in CODEfineTP 5-27
  - in CODisplayTP 5-42
- coPolarType parameter
  - in CODisplaySession 5-40
- coPrivType parameter
  - in CODEfineTP 5-27
  - in CODisplayTP 5-43
- COProcessSessionLimit routine 5-8
  - summarized 5-49
- coProgID parameter
  - in CODisplaySession 5-40



## **coQ**

coQueueBINDs parameter  
in CODEfineMode 5-21  
in CODisplayMode 5-35  
coQueueINITs parameter  
in CODEfineRemoteLU 5-24  
in CODisplayRemoteLU 5-35

## **coR**

coRcvPacing parameter  
in CODEfineMode 5-19  
in CODisplayMode 5-34  
coReinitType parameter  
in CODEfineMode 5-20  
in CODisplayMode 5-35  
COResetSessionLimit routine  
5-9 to 5-10  
summarized 5-50  
coRespType parameter  
in COChangeSessionLimit 5-5  
in COResetSessionLimit 5-9  
coRmtLUName parameter  
in COActivateSession 5-12  
in COChangeSessionLimit 5-4  
in CODEactivateSession 5-13  
in CODEfineMode 5-19  
in CODEfineRemoteLU 5-23  
in CODElete 5-30  
in CODisplayMode 5-33  
in CODisplayRemoteLU 5-37  
in CODisplaySession 5-39  
in COInitializeSessionLimit  
5-6  
in COProcessSessionLimit 5-8  
in COResetSessionLimit 5-9  
coRmtSecAcc parameter  
in CODisplayRemoteLU 5-38

## **coS**

coSecOp parameter  
in CODEfineLocalLU 5-16  
in CODEfineTP 5-28  
coSecReq parameter  
in CODEfineTP 5-28  
in CODisplayTP 5-43  
coSendPacing parameter  
in CODEfineMode 5-19  
in CODisplayMode 5-34  
coSessCrypt parameter  
in CODEfineMode 5-21  
in CODisplayMode 5-35  
coSessID parameter

in CODEactivateSession 5-13  
in CODisplaySession 5-39  
coSyncType parameter  
in CODEfineMode 5-20  
in CODEfineTP 5-29  
in CODisplayMode 5-35  
in CODisplayTP 5-44

## **coT**

coTermCount parameter  
in CODisplayMode 5-35  
coUserName parameter  
in CODEfineLocalLU 5-16  
in CODEfineTP 5-26  
in CODisplayLocalLU 5-32  
in CODisplayTP 5-42  
coUserProf parameter  
in CODEfineLocalLU 5-17  
in CODEfineTP 5-26  
in CODisplayLocalLU 5-32  
in CODisplayTP 5-42  
coUserPswd parameter  
in CODEfineLocalLU 5-17  
in CODisplayLocalLU 5-32  
coWaitTime parameter  
in CODEfineLocalLU 5-17  
in CODisplayLocalLU 5-32

## **cvB**

CVBackout routine 4-34

## **cvC**

cvConvID parameter  
in BCAllocate 4-42  
in BCGetAttributes 4-50  
in MCAllocate 4-12  
in MCGetAttributes 4-20  
cvConvType parameter  
in BCAllocate 4-40  
in CVGetType 4-35  
cvCVCBIndex parameter  
in CVWait 4-37  
cvCVCBList parameter  
in CVWait 4-37

## **cvD**

cvDataPtr parameter  
in BCDeallocate 4-46  
in BCReceiveAndWait 4-53  
in BCReceiveImmediate 4-55  
in BCSendData 4-58  
in BCSendError 4-60

in MCReceiveAndWait 4-23  
in MCReceiveImmediate 4-25  
in MCSendData 4-29  
cvDataSize parameter  
in BCDeallocate 4-46  
in BCPostOnReceipt 4-51  
in BCReceiveAndWait 4-53  
in BCReceiveImmediate 4-55  
in BCSendData 4-58  
in BCSendError 4-60  
in MCPostOnReceipt 4-21  
in MCReceiveAndWait 4-23  
in MCReceiveImmediate 4-25  
in MCSendData 4-29  
cvDeallocType parameter  
in BCDeallocate 4-45 to 4-46  
in MCDeallocate 4-15 to 4-16

## **cvE**

cvErrorType parameter  
in BCPostOnReceipt  
in BCReceiveAndWait  
in BCReceiveImmediate  
in BCSendError 4-60

## **cvF**

cvFillType parameter  
in BCPostOnReceipt 4-51  
in BCReceiveAndWait 4-53  
in BCReceiveImmediate 4-55  
cvFMHdrs parameter  
in MCSendData 4-30  
cvFullLclLUName parameter  
in BCGetAttributes 4-48  
in MCGetAttributes 4-18  
cvFullRmtLUName parameter  
in BCGetAttributes 4-48  
in MCGetAttributes 4-18

## **cvG**

CVGetType routine 4-35  
summarized 4-72

## **cvL**

cvLockType parameter  
in BCPrepareToReceive 4-52  
in MCPrepareToReceive 4-22  
cvLUWCorr parameter  
in BCGetAttributes 4-49  
in MCGetAttributes 4-19  
cvLUWID parameter  
in BCGetAttributes 4-49



- in MCGetAttributes 4-19
- cvLUWName parameter
  - in BCGetAttributes 4-49
  - in MCGetAttributes 4-19
- cvLUWSeq parameter
  - in BCGetAttributes 4-50
  - in MCGetAttributes 4-20

## cvM

- cvMapBuffPtr parameter
  - in MCAAllocate 4-10
- cvMapBuffSize parameter
  - in MCAAllocate 4-10
- cvMapName parameter
  - in MCReceiveAndWait 4-23
  - in MCReceiveImmediate 4-26
  - in MCSendDate 4-29
- cvMapProc parameter
  - in MCAAllocate 4-11
- cvModeName parameter
  - in BCAAllocate 4-40
  - in BCGetAttributes 4-49
  - in MCAAllocate 4-10
  - in MCGetAttributes 4-19

## cvP

- cvPIPBuffPtr parameter
  - in BCAAllocate 4-39
  - in MCAAllocate 4-10
- cvPIPBuffSize parameter
  - in BCAAllocate 4-40
  - in MCAAllocate 4-10
- cvPIPPtr parameter
  - in BCAAllocate 4-41
  - in MCAAllocate 4-12
- cvPIPSize parameter
  - in BCAAllocate 4-12
  - in MCAAllocate 4-12
- cvPIPUsed parameter
  - in BCAAllocate 4-41
  - in MCAAllocate 4-11
- cvPrepToRcvType parameter
  - in BCPrepareToReceive 4-52
  - in MCPPrepareToReceive 4-22
- cvProgID parameter
  - in BCGetAttributes 4-50
  - in MCGetAttributes 4-20

## cvR

- cvReqToSendRcvd parameter
  - in BCConfirm 4-43
  - in BCReceiveAndWait 4-54
  - in BCReceiveImmediate 4-56

- in BCSEndData 4-58
- in BCSEndError 4-60
- in MCConfirm 4-13
- in MCReceiveAndWait 4-24
- in MCReceiveImmediate 4-26
- in MCSendDate 4-30
- in MCSendError 4-31
- cvReturnCtl parameter
  - in BCAAllocate 4-41
  - in MCAAllocate 4-11
- cvRmtLUName parameter
  - in BCAAllocate 4-40
  - in BCGetAttributes 4-48
  - in MCAAllocate 4-10
  - in MCGetAttributes 4-18
- cvRmtProgName parameter
  - in BCAAllocate 4-40
  - in MCAAllocate 4-10

## cvS

- cvSecType parameter
  - in BCAAllocate 4-41 to 4-42
  - in MCAAllocate 4-12
- cvSenseData parameter
  - in BCSEndError 4-60
- CVSyncPoint routine 4-36
- cvSyncType parameter
  - in BCAAllocate 4-41
  - in BCGetAttributes 4-49
  - in MCAAllocate 4-11
  - in MCGetAttributes 4-19
- cvTestType parameter
  - in BCTest 4-62
  - in MCTest 4-32

## cvU

- cvUserName parameter
  - in BCAAllocate 4-40
  - in BCGetAttributes 4-49
  - in MCAAllocate 4-10
  - in MCGetAttributes 4-19
- cvUserProf parameter
  - in BCAAllocate 4-40
  - in BCGetAttributes 4-49
  - in MCAAllocate 4-11
  - in MCGetAttributes 4-19
- cvUserPswd parameter
  - in BCAAllocate 4-40
  - in MCAAllocate 4-10

## cvW

- CVWait routine 4-37
- summarized 4-72

- cvWhatRcvd parameter
  - in BCReceiveAndWait 4-53
  - in BCReceiveImmediate 4-55 to 4-56
  - in MCReceiveAndWait 4-23 to 4-24
  - in MCReceiveImmediate 4-26

## kA

- kAbendDealloc constant
  - in MCDeallocate 4-15
- kAbendProgDealloc constant
  - in BCDeallocate 4-45
- kAbendSvcDealloc constant
  - in BCDeallocate 4-45
- kAbendTimerDealloc constant
  - in BCDeallocate 4-45
- kAbortDetach constant
  - in TPDetach 7-8
- kAddParam constant
  - in CODEfineLocalLU 5-16
  - in CODEfineTP 5-28

## kB

- kBasicConv constant
  - in BCAAllocate 4-40
  - in CVGetType 4-35
- kBasicTPConv constant
  - in CODEfineTP 5-29
  - in CODisplayTP 5-44
- kBidderSess constant
  - in CODisplaySession 5-40
- kBothSecReq constant
  - in CODisplayTP 5-43
- kBufferFill constant
  - in BCPostOnReceipt 4-51
  - in BCReceiveAndWait 4-53
  - in BCReceiveImmediate 4-55

## kC

- kCleanupDeact constant
  - in CODEactivateSession 5-13
- kCNOSPriv constant
  - in CODEfineTP 5-27
  - in CODisplayTP 5-43
- kConfigWait constant
  - in TPAttach 7-6
- kConfirmDealloc constant
  - in BCDeallocate 4-45
  - in MCDeallocate 4-15
- kConfirmDeallocRcvd constant
  - in BCReceiveAndWait 4-54
  - in BCReceiveImmediate 4-56



- in MCRReceiveAndWait 4-24
- in MCRReceiveImmediate 4-26
- kConfirmModeSync constant
  - in CODEfineMode 5-20
  - in CODisplayMode 5-35
- kConfirmRcv constant
  - in BCPrepareToReceive 4-52
- kConfirmRcvd constant
  - in BCReceiveAndWait 4-54
  - in BCReceiveImmediate 4-56
  - in MCRReceiveAndWait 4-24
  - in MCRReceiveImmediate 4-26
- kConfirmSendRcvd constant
  - in BCReceiveAndWait 4-54
  - in MCRReceiveAndWait 4-24
  - in MCRReceiveImmediate 4-26
- kConfirmSync constant
  - in BCallocate 4-41
  - in BCGetAttributes 4-49
  - in MCAAllocate 4-11
  - in MCGetAttributes 4-19
- kConfirmTPSync constant
  - in CODEfineTP 5-29
  - in CODisplayTP 5-44
- kConnectDial constant
  - in NOActivateStation 6-6
- kConvSecAcc constant
  - in CODEfineRemoteLU 5-24
  - in CODisplayRemoteLU 5-38
- kConvSecReq constant
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43

## **KD**

- kDataComplRcvd constant
  - in BCReceiveAndWait 4-53
  - in BCReceiveImmediate 4-55
  - in MCRReceiveAndWait 4-23
  - in MCRReceiveImmediate 4-26
- kDataIncomplRcvd constant
  - in BCReceiveAndWait 4-54
  - in BCReceiveImmediate 4-56
  - in MCRReceiveAndWait 4-24
  - in MCRReceiveImmediate 4-26
- kDataRcvd constant
  - in BCReceiveAndWait 4-53
  - in BCReceiveImmediate 4-55
- kDataTruncRcvd constant
  - in BCReceiveImmediate 4-56
- kDefinePriv constant
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43
- kDelayAllocReturn constant
  - in BCallocate 4-41
  - in MCAAllocate 4-11

- kDeleteParam constant
  - in CODEfineLocalLU 5-16
  - in CODEfineRemoteLU 5-23, 5-24
  - in CODEfineTP 5-26, 5-28
- kDevelMsgsQClass constant
  - in NODefineMessageQueue 6-12
  - in NODisplayMessageQueue 6-15
- kDevelMsgsQSev constant
  - in NODefineMessageQueue 6-13
  - in NODisplayMessageQueue 6-16
- kDiagMsgsQType constant
  - in NODefineMessageQueue 6-12
  - in NODisplayMessageQueue 6-15
- kDialInOffDial constant
  - in NOdeactivateStation 6-10
- kDialInOnDial constant
  - in NOActivateStation 6-6
- kDisablePermTP constant
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43
- kDisableTempTP constant
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43
- kDisconnectDial constant
  - in NOdeactivateStation 6-10
- kDisplayPriv constant
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43

## **KE**

- kEitherLUInit constant
  - in CODEfineMode 5-20
  - in CODisplayMode 5-35
- kEitherTPConv constant
  - in CODEfineTP 5-29
  - in CODisplayTP 5-44
- kEnableTP constant
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43
- kErrorMsgsQSev constant
  - in NODefineMessageQueue 6-13
  - in NODisplayMessageQueue 6-16
- kErrorMsgsQType constant
  - in NODefineMessageQueue 6-12

- in NODisplayMessageQueue 6-15

## **kF**

- kFirstSpkrSess constant
  - in CODisplaySession 5-40
- kFlushDealloc constant
  - in BCDeallocate 4-45
  - in MCDeallocate 4-15
- kFlushRcv constant
  - in BCPrepareToReceive 4-52
- kFMHDataComplRcvd constant
  - in MCRReceiveAndWait 4-24
  - in MCRReceiveImmediate 4-26
- kFMHDataIncomplRcvd constant
  - in MCRReceiveAndWait 4-24
  - in MCRReceiveImmediate 4-26
- kFMHDataTruncRcvd constant
  - in MCRReceiveAndWait 4-24
  - in MCRReceiveImmediate 4-26
- kFuncNotSupp constant
  - in CODEfineLocalLU 5-17
  - in CODEfineMode 5-21
  - in CODEfineRemoteLU 5-24
  - in CODEfineTP 5-27, 5-29
  - in CODisplayLocalLU 5-32
  - in CODisplayMode 5-35
  - in CODisplayRemoteLU 5-38
  - in CODisplayTP 5-42, 5-43
  - in NODefineNode 6-24
  - in NODisplayNode 6-32
- kFuncSupp constant
  - in CODEfineLocalLU 5-17
  - in CODEfineMode 5-21
  - in CODEfineRemoteLU 5-24
  - in CODEfineTP 5-27, 5-29
  - in CODisplayLocalLU 5-32
  - in CODisplayMode 5-35
  - in CODisplayRemoteLU 5-38
  - in CODisplayTP 5-42, 5-43
  - in NODefineNode 6-24
  - in NODisplayNode 6-32

## **kl**

- kIgnoreParam constant
  - in CODEfineLocalLU 5-16, 5-17
  - in CODEfineMode 5-20, 5-21
  - in CODEfineRemoteLU 5-23, 5-24
  - in CODEfineTP 5-26, 5-27, 5-28, 5-29



- in CODisplayLocalLU 5-31, 5-32
- in CODisplayMode 5-34, 5-35
- in CODisplayRemoteLU 5-37
- in CODisplaySession 5-39
- in CODisplayTP 5-41, 5-42
- in NODefineLine 6-21, 6-22
- in NODefineNode 6-23, 6-24
- in NODisplayCP 6-28
- in NODisplayLine 6-29
- in NODisplayStation 6-33
- kImmedAllocReturn constant
  - in BCallocate 4-41
  - in MCallocate 4-11
- kIncomplMode constant
  - for mcpbRcvMode 4-6
- kInfoMsgsQType constant
  - in NODefineMessageQueue 6-12
  - in NODisplayMessageQueue 6-15

## KL

- kLineActive constant
  - in NODisplayLine 6-30
- kLinePendActive constant
  - in NODisplayLine 6-30
- kLinePendReset constant
  - in NODisplayLine 6-30
- kLineReset constant
  - in NODisplayLine 6-30
- kLLFill constant
  - in BCPostOnReceipt 4-51
  - in BCReceiveAndWait 4-53
  - in BCReceiveImmediate 4-55
- kLLTruncRcvd constant
  - in BCReceiveAndWait 4-54
  - in MCreceiveAndWait 4-24
  - in MCreceiveImmediate 4-26
- kLocalDealloc constant
  - in BCDeallocate 4-46
  - in MCDeallocate 4-16
- kLogMsgsQClass constant
  - in NODefineMessageQueue 6-12
  - in NODisplayMessageQueue 6-15
- kLongLock constant
  - in BCPPrepareToReceive 4-52
- kLowLevelInfoMsgsQSev constant
  - in NODefineMessageQueue 6-13
  - in NODisplayMessageQueue 6-16

- kLUAttach constant
  - in TPAttach 7-6

## KM

- kMappedConv constant
  - in BCallocate 4-40
  - in CVGetType 4-35
- kMappedTPConv constant
  - in CODisplayTP 5-44
- kMaxWait constant
  - in TPAttach 7-6

## KN

- kNextEntry constant
  - in CODisplayLocalLU 5-31, 5-32
  - in CODisplayMode 5-34
  - in CODisplayRemoteLU 5-37
  - in CODisplaySession 5-39
  - in CODisplayTP 5-41, 5-42
  - in NODisplayCP 6-28
  - in NODisplayLine 6-29
  - in NODisplayStation 6-33
- kNoChangeQClass constant
  - in NODefineMessageQueue 6-12
- kNoChangeQSev constant
  - in NODefineMessageQueue 6-13
- kNoChangeQType constant
  - in NODefineMessageQueue 6-12
- kNodeOperMsgsQClass constant
  - in NODefineMessageQueue 6-12
  - in NODisplayMessageQueue 6-15
- kNoPriv constant
  - in CODEfineTP 5-27
  - in CODisplayTP 5-43
- kNormalDeact constant
  - in CODEactivateSession 5-13
- kNormalDetach constant
  - in TPDetach 7-8
- kNormalInfoMsgsQSev constant
  - in NODefineMessageQueue 6-13
  - in NODisplayMessageQueue 6-16
- kNoSecAcc constant
  - in CODEfineRemoteLU 5-24
  - in CODisplayRemoteLU 5-38
- kNoSec constant
  - in BCallocate 4-41

- kNoSecReq constant
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43
- kNoSync constant
  - in BCallocate 4-41
  - in BCGetAttributes 4-49
  - in MCallocate 4-11
  - in MCGetAttributes 4-19
- kNotifMsgsQType constant
  - in NODefineMessageQueue 6-12
  - in NODisplayMessageQueue 6-15
- kNoTPSync constant
  - in CODEfineTP 5-29
  - in CODisplayTP 5-44

## KO

- kOperInit constant
  - in CODEfineMode 5-20
  - in CODisplayMode 5-35

## KP

- kPostTest constant
  - in BCTest 4-62
  - in MCTest 4-32
- kPriLUInit constant
  - in CODEfineMode 5-20
  - in CODisplayMode 5-35
- kProfSecReq constant
  - in CODEfineTP 5-28
  - in CODisplayTP 5-43
- kProgError constant
  - in BCSEndError 4-60
- kProgErrorsQSev constant
  - in NODefineMessageQueue 6-13
  - in NODisplayMessageQueue 6-16
- kProgSec constant
  - in BCallocate 4-42

## KR

- kRcvMapping constant
  - for mcpbMapCmd 4-6
- kReplaceParam constant
  - in CODEfineLocalLU 5-16
- KReplaceParam constant
  - in CODEfineRemoteLU 5-23, 5-24
- kReplaceParam constant
  - in CODEfineTP 5-26
- kReqToSendTest constant



in BCTest 4-62  
in MCTest 4-32

## **kS**

kSameSec constant  
in BCallocate 4-42  
kSDLC4800 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLC9600 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLC19200 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLC300 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLC1200 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLC2400 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCAccess constant  
in NODefineNode 6-23  
in NODisplayNode 6-32  
kSDLCFullDuplex constant  
in NODefineLine 6-22  
in NODisplayLine 6-31  
kSDLCHalfDuplex constant  
in NODefineLine 6-22  
in NODisplayLine 6-31  
kSDLCLeased constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCLine constant  
in NODefineLine 6-20  
in NODisplayLine 6-30  
kSDLCLine4 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCLine1 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCLine3 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCLine2 constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCMultiPoint constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCNegotiable constant

in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCNRZ constant  
in NODefineLine 6-22  
in NODisplayLine 6-31  
kSDLCNRZI constant  
in NODefineLine 6-22  
in NODisplayLine 6-31  
kSDLCPrimary constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCSecondary constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSDLCSwitched constant  
in NODefineLine 6-21  
in NODisplayLine 6-30  
kSecLUInit constant  
in CODefineMode 5-20  
in CODisplayMode 5-35  
kSendMapping constant  
for mcpbMapCmd 4-6  
kSendRcvd constant  
in BCReceiveAndWait 4-54  
in BCReceiveImmediate 4-56  
in MCReceiveAndWait 4-24  
in MCReceiveImmediate 4-26  
kSessCtlPriv constant  
in CODefineTP 5-27  
in CODisplayTP 5-43  
kShortLock constant  
in BCPrepareToReceive 4-52  
kSrcResp constant  
in COChangeSessionLimit 5-5  
in COResetSessionLimit 5-9  
kSrvrAttach constant  
in TPAttach 7-6  
kStationActive constant  
in NODisplayStation 6-34  
kStationPendCont constant  
in NODisplayStation 6-34  
kStationPendReset constant  
in NODisplayStation 6-34  
kStationPendResp constant  
in NODisplayStation 6-34  
kStationReset constant  
in NODisplayStation 6-34  
kStationResetPendResp  
constant  
in NODisplayStation 6-34  
kSvcError constant  
in BCSendError 4-60  
kSvcTPPriv constant  
in CODefineTP 5-28  
in CODisplayTP 5-43

kSyncDealloc constant  
in BCDeallocate 4-45  
in MCDeallocate 4-15  
kSyncPtModeSync constant  
in CODefineMode 5-20  
in CODisplayMode 5-35  
kSyncPtSync constant  
in BCallocate 4-41  
in BCGetAttributes 4-49  
in MCallocate 4-11  
kSyncPtTPSync constant  
in CODefineTP 5-29  
in CODisplayTP 5-44  
in MCGetAttributes 4-19

## **kT**

kTgtResp constant  
in COChangeSessionLimit 5-5  
in COResetSessionLimit 5-9  
kTraceMsgsQClass constant  
in NODefineMessageQueue  
6-12  
in NODisplayMessageQueue  
6-15  
kTruncMode constant  
for mcpbRcvMode 4-6

## **kU**

kUserNameSecReq constant  
in CODefineTP 5-28  
in CODisplayTP 5-43  
kVerifSecAcc  
in CODefineRemoteLU 5-24  
kVerifSecAcc constant  
in CODisplayRemoteLU 5-38

## **kW**

kWaitAttach constant  
in TPAttach 7-6  
kWhenAllocReturn constant  
in BCallocate 4-41  
in MCallocate 4-11

## **m**

MCallocate routine 4-9 to 4-12  
summarized 4-67  
MCCConfirmed routine 4-14  
summarized 4-68  
MCCConfirm routine 4-13  
summarized 4-67  
MCDeallocate routine 4-15 to  
4-16



- summarized 4-68
- mcDupMapNameErr constant
  - for mcpbResult 4-6
- mcErr constant
  - for mcpbResult 4-6
- MCFlush routine 4-17
  - summarized 4-68
- MCGetAttributes routine 4-18
  - to 4-20
  - summarized 4-68 to 4-69
- mcMapNameErr constant
  - for mcpbResult 4-6
- mcNoErr constant
  - for mcpbResult 4-6
- MCPostOnReceipt routine 4-21
  - summarized 4-69
- MCPrepareToReceive routine 4-22
  - summarized 4-69
- MCReceiveAndWait routine 4-23 to 4-24
  - summarized 4-69
- MCReceiveImmediate routine 4-25 to 4-27
  - summarized 4-70
- MCRequestToSend routine 4-28
  - summarized 4-70
- MCSendData routine 4-29 to 4-30
  - summarized 4-70
- MCSendError routine 4-31
  - summarized 4-70 to 4-71
- MCTest routine 4-32
  - summarized 4-71

## noA

- noAccessType parameter
  - in NODEfineNode 6-23
  - in NODisplayNode 6-32
- NOActivateLine routine 6-3
  - summarized 6-39
- NOActivateLU routine 6-4
  - summarized 6-39
- NOActivateNode routine 6-5
  - summarized 6-39
- NOActivateStation routine 6-6
  - summarized 6-39
- noALSAddr parameter
  - in NODEfineStation 6-26
  - in NODisplayStation 6-34
- noALSName parameter
  - in NOActivateStation 6-6
  - in NODEactivateStation 6-10
  - in NODEfineStation 6-25

- in NODElete 6-27
- in NODisplayStation 6-33
- noALSStatus parameter
  - in NODisplayStation 6-34

## noC

- noCorrID parameter
  - in NOActivateLine 6-3
  - in NOActivateLU 6-4
  - in NOActivateNode 6-5
  - in NOActivateStation 6-6
  - in NODEactivateLine 6-7
  - in NODEactivateLU 6-8
  - in NODEactivateNode 6-9
  - in NODEactivateStation 6-10
  - in NODisplayMessage 6-14
- noCPName parameter
  - in NODEfineCP 6-18
  - in NODEfineStation 6-25
  - in NODElete 6-27
  - in NODisplayCP 6-28
  - in NODisplayStation 6-33
- noCUID parameter
  - in NODEfineCP 6-19
  - in NODisplayCP 6-28

## noD

- noDataPtr parameter
  - in NODisplayMessage 6-14
- noDataSize parameter
  - in NODisplayMessage 6-14
- NODEactivateLine routine 6-7
  - summarized 6-40
- NODEactivateLU routine 6-8
  - summarized 6-40
- NODEactivateNode routine 6-9
  - summarized 6-40
- NODEactivateStation routine 6-10
  - summarized 6-40
- NODEfineCP routine 6-18 to 6-19
  - summarized 6-42
- NODEfineLine routine 6-20 to 6-22
  - summarized 6-42
- NODEfineMessageQueue routine 6-12 to 6-13
  - summarized 6-41
- NODEfineNode routine 6-23 to 6-24
  - summarized 6-42 to 6-43
- NODEfineStation routine 6-25 to 6-26

- summarized 6-43
- NODElete routine 6-27
  - summarized 6-43
- noDialType parameter
  - in NOActivateStation 6-6
  - in NODEactivateStation 6-10
- NODisplayCP routine 6-28
  - summarized 6-43
- NODisplayLine routine 6-29 to 6-31
  - summarized 6-43 to 6-44
- NODisplayMessageQueue routine 6-15 to 6-16
  - summarized 6-41
- NODisplayMessage routine 6-14
  - summarized 6-41
- NODisplayNode routine 6-32
  - summarized 6-44
- NODisplayStation routine 6-33 to 6-34
  - summarized 6-44

## noE

- noExchID parameter
  - in NODEfineCP 6-18
  - in NODEfineNode 6-23
  - in NODisplayCP 6-28
  - in NODisplayNode 6-32

## noL

- noLclLUName parameter
  - in NOActivateLU 6-4
  - in NODEactivateLU 6-8
- noLineName parameter
  - in NOActivateLine 6-3
  - in NODEactivateLine 6-7
  - in NODEfineLine 6-20
  - in NODEfineStation 6-25
  - in NODElete 6-27
  - in NODisplayLine 6-29
  - in NODisplayStation 6-33
- noLinePtr parameter
  - in NODEfineLine 6-20
  - in NODisplayLine 6-29
- noLineStatus parameter
  - in NODisplayLine 6-30
- noLineType parameter
  - in NODEfineLine 6-20
  - in NODisplayLine 6-30
- noLogMsgs parameter
  - in NODEfineNode 6-24
  - in NODisplayNode 6-32



## **noM**

noMonTimer parameter  
in NODefineNode 6-23  
in NODisplayNode 6-32

## **noN**

noNextALSName parameter  
in NODisplayStation 6-33  
noNextCPName parameter  
in NODisplayCP 6-28  
noNextLineName parameter  
in NODisplayLine 6-29  
noNodeMsgs parameter  
in NODefineNode 6-24  
in NODisplayNode 6-32

## **noP**

noPhoneNumber parameter  
in NODefineStation 6-25  
in NODisplayStation 6-33

## **noQ**

noQueueClass parameter  
in NODefineMessageQueue 6-12  
in NODisplayMessageQueue 6-15  
noQueueEnable parameter  
in NODefineMessageQueue 6-12  
in NODisplayMessageQueue 6-15  
noQueueName parameter  
in NODefineMessageQueue 6-12  
in NODisplayMessage 6-14  
in NODisplayMessageQueue 6-15  
noQueueSev parameter  
in NODefineMessageQueue 6-13  
in NODisplayMessageQueue 6-15  
noQueueType parameter  
in NODefineMessageQueue 6-12  
in NODisplayMessageQueue 6-15

## **noS**

noStopSrvr parameter  
in NODeactivateNode 6-9

## **noW**

noWaitForMsg parameter  
in NODisplayMessage 6-14

## **O**

OpenDriver routine 3-2

## **P**

PBOpen routine 3-2

## **sd**

sdlcConnType parameter  
in NODefineLine 6-21  
in NODisplayLine 6-30  
sdlcDuplexType parameter  
in NODefineLine 6-22  
in NODisplayLine 6-30  
sdlcIdleTime parameter  
in NODefineLine 6-22  
in NODisplayLine 6-31  
sdlcLineNum parameter  
in NODefineLine 6-21  
in NODisplayLine 6-30  
sdlcLineSpeed parameter  
in NODefineLine 6-21  
in NODisplayLine 6-30  
sdlcMaxBTU parameter  
in NODefineLine 6-21  
in NODisplayLine 6-30  
sdlcMaxIframe parameter  
in NODefineLine 6-22  
in NODisplayLine 6-30  
sdlcMaxRetry parameter  
in NODefineLine 6-22  
in NODisplayLine 6-30  
sdlcNPRcvTime parameter  
in NODefineLine 6-22  
in NODisplayLine 6-31  
sdlcNRZIType parameter  
in NODefineLine 6-22  
in NODisplayLine 6-30  
sdlcRoleType parameter  
in NODefineLine 6-21  
in NODisplayLine 6-30

## **TP**

TPAsciiToEbcDic routine 7-10  
TPAttach routine 7-5 to 7-7  
tpAttachType parameter  
in TPAttach 7-6  
tpConvID parameter  
in TPAttach 7-7  
tpCVCBPtr parameter  
in TPAttach 7-6  
tpDataPtr parameter  
in TPAsciiToEbcDic 7-10  
TPEbcDicToAscii 7-11  
tpDataSize parameter  
TPAsciiToEbcDic 7-10  
TPEbcDicToAscii 7-11  
TPDetach routine 7-8  
tpDetachType parameter  
in TPDetach 7-8  
TPEbcDicToAscii routine 7-11  
tpLclLUName parameter  
in TPAttach 7-6  
tpLclProgName parameter  
in TPAttach 7-6  
tpMapBuffPtr parameter  
in TPAttach 7-6  
tpMapBuffSize parameter  
in TPAttach 7-6  
tpMapProc parameter  
in TPAttach 7-6  
tpPIPBuffPtr parameter  
in TPAttach 7-6  
tpPIPBuffSize parameter  
in TPAttach 7-6  
tpPIPPtr parameter  
in TPAttach 7-7  
tpPIPSize parameter  
in TPAttach 7-7  
tpProgID parameter  
in TPAttach 7-7  
tpSrvrEntityPtr parameter  
in TPAttach 7-2, 7-6  
tpTPCBPtr parameter  
in TPAttach 7-5  
in TPDetach 7-8  
tpWaitTime parameter  
in TPAttach 7-6







## THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using the Apple Macintosh™ Plus and Microsoft® Word. Proof fipages were created on the Apple LaserWriter® Plus, and final pages were created on the Varityper VT600. POSTSCRIPT™, the LaserWriter's page-description language, was developed by Adobe Systems Incorporated.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Program listings are set in Apple Courier, a monospaced font.









MacAPPC™

# Technical Note

## MacAPPC on leased SDLC multi-point lines

One of the more popular configurations used in MacAPPC™ Beta testing was *leased multi-point SDLC connections*. This technical note augments the Apple® MacAPPC Programmer's Reference and User's Guide in detailing the setup of MacAPPC servers in these configurations.

---

---

### Introduction

When IBM developed Synchronous Data Link Control (SDLC), peer-to-peer communication was not a design criteria. Although this has been addressed with extensions to the architecture applied in a point-to-point environment, there are some rules to follow and some restrictions on session support in the SDLC multi-point environment.

---

---

### Primary and secondary nodes on a multi-point circuit

The primary-secondary relationship of Physical Units on a multi-point line is still with us. On multi-point SDLC lines, there **MUST** be one and only one primary station. This station controls the line's master modem, and is in essence in control of the secondary stations on the line.

If the primary station is a Macintosh® with MacAPPC, *the LINE definition must be optioned as Primary and Multi*. The secondary stations' *LINE definition must be optioned as Secondary and Lease* whenever a Macintosh with MacAPPC is used as a secondary station (see Figure 1). Always use *half duplex* on a multi-point SDLC circuit.



Line: LINEONE	
Line Name:	LINEONE
Line Type:	SDLC
-----	
Line Number:	<input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4
Role Type:	<input checked="" type="radio"/> Prim <input type="radio"/> Sec <input type="radio"/> Negot
Connect Type:	<input type="radio"/> Lease <input checked="" type="radio"/> Multi <input type="radio"/> Switch
Max BTU:	265
Line Speed:	<input type="radio"/> 300 <input type="radio"/> 1200 <input type="radio"/> 2400 <input type="radio"/> 4800 <input checked="" type="radio"/> 9600 <input type="radio"/> 19200
Max Retries:	3
Idle Time:	800
NP Recv Time:	10000
Max I-Frames:	7
NRZI Support:	<input checked="" type="radio"/> NRZ <input type="radio"/> NRZI
Duplex Type:	<input checked="" type="radio"/> Half <input type="radio"/> Full

Line: LINEONE	
Line Name:	LINEONE
Line Type:	SDLC
-----	
Line Number:	<input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4
Role Type:	<input type="radio"/> Prim <input checked="" type="radio"/> Sec <input type="radio"/> Negot
Connect Type:	<input checked="" type="radio"/> Lease <input type="radio"/> Multi <input type="radio"/> Switch
Max BTU:	265
Line Speed:	<input type="radio"/> 300 <input type="radio"/> 1200 <input type="radio"/> 2400 <input type="radio"/> 4800 <input checked="" type="radio"/> 9600 <input type="radio"/> 19200
Max Retries:	3
Idle Time:	800
NP Recv Time:	10000
Max I-Frames:	7
NRZI Support:	<input checked="" type="radio"/> NRZ <input type="radio"/> NRZI
Duplex Type:	<input checked="" type="radio"/> Half <input type="radio"/> Full

**Figure 1**  
Primary Line Definition and Secondary Line Definition

To assist in troubleshooting multi-point problems, the following is a list of symptoms indicating the attempted use of full-duplex definitions in this environment:

- ☐ The first slave modem on the circuit will show a constant transmit.
- ☐ The first slave modem will raise request to send and keep it raised.
- ☐ If the modem has an anti-streaming feature, it will be activated.
- ☐ You will be unable to activate the logical units.
- ☐ Transaction Programs will fail with appcHResult=005 and appcLOResult=009 on allocate statements.

Insure your modems are properly strapped for multi-point operation by referring to the user manual supplied by the leased line modem vendor. Some of these modems can have complex timing and strapping requirements, especially if the line is a tail circuit off of a multiplexed point to point circuit. In IBM networks, such tail circuits are quite commonplace, with modem timing being a critical factor. When in doubt, contact the modem vendor's technical support organization to insure proper modem strapping.

---



---

## Multi-point session SDLC restrictions

Using SDLC, *Secondaries can ONLY communicate and establish sessions with the primary station and vice versa.* Unless you have a routing transaction program running in the primary station to redirect traffic, secondaries cannot communicate with each other. In a mainframe oriented environment, this is not as bad as it sounds. Most LU 6.2 multi-point circuits connect to a mainframe computer which acts as the primary, although with MacAPPC a Macintosh can play this primary role.



## ALS addressing on multi-point circuits

The role of the ALS parameter used to handle addressing changes with the use of SDLC multi-point circuits. This is another critical area for proper operation in this environment. If your primary station expects to see secondary stations addressed as C2 and C3 on a multi-point line, and your primary is a Macintosh, there must be separate PARTNER definitions for each station with the ALS ADDRESS fields set at C2 and C3 (see Figure 2).

Partner: PARTNER2	
Partner Name:	PARTNER2
Line Name:	LINEONE
<input checked="" type="radio"/> Exch ID:	FFF00002
<input type="radio"/> CPU ID:	
-----	
ALS Address:	C2
Phone Number:	

Partner: PARTNER3	
Partner Name:	PARTNER3
Line Name:	LINEONE
<input checked="" type="radio"/> Exch ID:	FFF00003
<input type="radio"/> CPU ID:	
-----	
ALS Address:	C3
Phone Number:	

**Figure 2**  
Primary's C2 Partner Definition and Primary's C3 Partner Definition

The secondary stations must know their own addresses in order to respond to packets sent to them by the primary. This is identified to the secondary station in *Secondary/Lease mode* is by using the *ALS ADDRESS in the PARTNER definition to define what address the secondary will respond to*. Using the above example, the unit you wish to be identified as C2 would have C2 in this field (see Figure 3), and the C3 unit would use C3 in this field. Secondary units **MUST** have unique addresses, and these addresses **MUST** be defined to the primary (as outlined earlier) in order to establish communications.

Partner: PARTNER1	
Partner Name:	PARTNER1
Line Name:	LINEONE
<input checked="" type="radio"/> Exch ID:	FFF00001
<input type="radio"/> CPU ID:	
-----	
ALS Address:	C2
Phone Number:	

Partner: PARTNER1	
Partner Name:	PARTNER1
Line Name:	LINEONE
<input checked="" type="radio"/> Exch ID:	FFF00001
<input type="radio"/> CPU ID:	
-----	
ALS Address:	C3
Phone Number:	

**Figure 3**  
Secondary Station C2's Partner Definition and Secondary Station C3's Partner Definition



---

---

## XIDs on multi-point circuits

XIDs are also important in SDLC multi-point lines. Each station must have a unique XID. This is defined in the Node definition for each unit. Using the example, assign FFF00001 to the primary, FFF00002 to secondary station C2, and FFF00003 to secondary station C3. If the primary station is a Macintosh, the PARTNER definitions should reflect the *XID's of the secondary units* (see Figure 2). In the secondary units, the PARTNER definition should always contain the *XID of the Primary* (see Figure 3).

Apple, the Apple logo, and Macintosh are registered trademarks, and MacAPPC is a trademark of Apple Computer, Inc.