APPC Networking Services for Windows

SC31-8126-00

Application Programmer's Reference

Version 1.0

IBM

APPC Networking Services for Windows

SC31-8126-00

Application Programmer's Reference

Version 1.0

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vii.

First Edition (November 1994)

This edition applies to Version 1.0 of the licensed program IBM APPC Networking Services for Windows, program number 5871-AAA, feature number 9737, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation
Networking Services for Windows
Department F94
P.O. Box 12195
Research Triangle Park, North Carolina 27709-2195
U.S.A.

When you send information to IBM, you grant IBM a non-exclusive right to use and distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1994. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices vii
Trademarks and Service Marks viii

About This Book xi
Who Should Read This Book xi
Conventions Used in This Book xii
Terms Used in This Book xii

Part 1. Introduction

Chapter 1. Introduction 1-1

Part 2. Writing Distributed Applications to the CPI-C API

Chapter 2. Using CPI-C Calls in Networking Services for Windows 2-1
Overview of CPI-C 2-1
Support of CPI-C Conformance Classes 2-2
Mandatory Conformance Classes 2-2
Optional Conformance Classes Supported 2-2
Optional Conformance Classes Not Supported 2-2
Non-Applicable CPI-C Terms and Concepts 2-3
CPI-C Calls Supported 2-4
Pseudonym Files 2-6
Examples of Using C 2-6
CPI-C Function Calls in C 2-6
Using the Pseudonym Files in C Language Programs 2-6
Using Other Languages 2-7
Linking With the CPI-C Import Library 2-7
Memory Considerations 2-7
Data Buffers 2-7
Stack Size 2-7
Defining Side Information 2-8
Usage Notes for Mode_Name and TP_Name 2-8
Mode_Name 2-8
Restrictions on Transaction Program Names 2-8
Multiple CMACCPs 2-8
How Dangling Conversations Are Deallocated 2-9
Diagnosing Errors 2-9
Log_Data 2-9
Identifying Product-Specific Errors 2-9
Deviations from the CPI Communications Architecture 2-11
Return_control Characteristic for Allocate (CMALLC) 2-11
CM_PROGRAM_PARAMETER_CHECK Return Code 2-11
Log Data Support 2-12

© Copyright IBM Corp. 1994

Chapter 3. Designing CPI-C Applications 3-1
Choosing an Application Model 3-1
The Client/Server Application Model 3-2
The Transaction Processing Application Model 3-3
The Distributed Application Model 3-3
The Real-Time Control Model 3-4
Combining CPI-C Calls to Perform Transactions 3-5
The Pipeline Transaction (One Way Bracket) 3-6
The Inquiry Transaction (Request and Reply) 3-7
The Credit-Check Transaction (Confirmed Delivery) 3-8
The File-Transfer Transaction (Batch Send) 3-10
The Database-Update Transaction (Conversational Reply) 3-11
Starting a Program 3-12
Choosing a Conversation Type 3-12
Implementing a Conversation 3-14
Allocating a Conversation to a Partner Application 3-15
Accepting a Conversation Started by a Partner Application 3-15
Deallocating a Conversation 3-15
Sending Data 3-15
Forcing Data to be Transmitted 3-15
Receiving Data 3-15
Granting Permission to Send 3-16
Requesting Permission to Send 3-16
Reporting an Error 3-16
Requesting Confirmation 3-17
Accepting or Rejecting a Confirmation Request 3-17
Extracting Conversation Information 3-17
Designing CPI-C Applications to Improve Performance 3-17

Chapter 4. Avoiding Deadlocks and Wait Conditions 4-1
Understanding Deadlock Situations 4-2
Understanding CPI-C Wait Conditions 4-2
Waiting for Data 4-2
Waiting for an Incoming Attach 4-3
Waiting for Pacing 4-3
Waiting for Confirmation 4-4
Waiting after a Deallocate or Send_Error Call 4-4
Identifying a CPI-C Application Deadlock 4-5
Stopping a Deadlocked CPI-C Program 4-5

Chapter 5. Managing Incoming Attaches 5-1
What Is the Attach Manager? 5-2
How Does an Incoming Conversation Start? 5-2
What Information Does the Attach Manager Need? 5-3
Processing Incoming Attaches 5-6
Adding Incoming Attaches to the Attach List 5-6
Calling the Program Launcher 5-7
Removing an Attach from the Attach List 5-7

Rejecting Incoming Attaches 5-8
Processing Accepts 5-8
Starting Programs and the Program Launcher 5-12
Starting Programs 5-12
Starting and Stopping the Program Launcher 5-12

Part 3. Using the Node Operator Facility Application Program Interface

Chapter 6. Using Networking Services Node Operator Application Program Interface 6-1
Verb Data Structures 6-1
Return Values and Fields 6-2
Functional Value 6-2
Other Returned Error Indicators 6-3
Other Returned Parameters 6-4
Define Symbolic Constants 6-4
NSACT 6-6
Allocate Return Codes 6-8
NSATTACH 6-10
NSCONLAN 6-11
NSCONMPT 6-13
NSCONREM 6-14
NSCONTWX 6-16
NSDEACT 6-17
NSDEASES 6-18
NSDEFMO 6-19
NSDELMO 6-20
NSDEFSI 6-21
NSDELSI 6-23
NSDEFTP 6-24
NSDELTP 6-26
NSDISCON 6-27
NSINFO 6-28
NSREJATT 6-32
NSSTART 6-33
NSSTOP 6-34

Part 4. Appendixes, Glossary, and Index

Appendix A. Converting Networking Services/DOS Applications to Networking Services for Windows A-1

Appendix B. Getting Information from IBM's APPC Market Enablement Group B-1
APPC Developer Assistance Program (DAP) B-1
Other Services of APPC Market Enablement B-4
Get the Latest Information on CPI-C, APPC, & APPN Online B-4

Internet B-4
OS/2 BBS B-4
CompuServe - the APPC/APPN Forum (GO APPC) B-4

Bibliography X-1
Softcopy X-1
Publications X-1
Where to Find Networking Services Information X-1
Where to Find Related Information X-1
Configuration Information for Other Platforms X-2

Glossary X-3

Index X-15

Notices

The licensed programs and related licensed material described herein are provided by IBM under the IBM Program License Agreement (PLA).

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. **This disclaimer does not apply in the United Kingdom or elsewhere if inconsistent with local law.**

This publication may include inaccuracies or errors. IBM may change this publication, the product described herein, or both.

References in this publication to IBM products or services do not imply that they will be available everywhere IBM operates, nor that only IBM products or services can be used. Functionally equivalent products or services that do not infringe IBM's legal rights can be used instead. Operation with products or services other than those expressly designated by IBM is your responsibility.

IBM may have patents or pending patent applications covering subject matter described herein. This document neither grants nor implies any license or immunity under any IBM or third-party patents, patent applications, trademarks, copyrights, or other similar rights, or any right to refer to IBM in any marketing activities. Other than responsibilities assumed via the IBM Customer Agreement, IBM assumes no responsibility for any infringement of third-party rights that may result from use of the subject matter disclosed in this publication or from the manufacture, use, lease, or sale of machines or programs described herein.

Licenses under IBM's utility patents are available on reasonable and nondiscriminatory terms. IBM does not grant licenses under its appearance design patents. Direct licensing inquiries in writing to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, U.S.A.

For online versions of this book, we authorize you to:

Copy, modify, and print the documentation contained on the media, for use within your enterprise, provided you reproduce the copyright notice, all warning statements, and other required statements on each copy or partial copy.

Transfer the original unaltered copy of the documentation when you transfer the related IBM product (which may be either machines you own, or programs, if the program's license terms permit a transfer). You must, at the same time, destroy all other copies of the documentation.

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization.

© Copyright IBM Corp. 1994

THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Your failure to comply with the terms above terminates this authorization. Upon termination, you must destroy your machine readable documentation.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) on their first occurrences in this publication, are trademarks or service marks of the IBM Corporation in the United States or other countries:

AIX

AS/400

BookManager

C/2

CICS

CICS/ESA

IBM

IBMLink

IIN

Library Reader

OS/2

VTAM.

The following terms, denoted by a double asterisk (**) on their first occurrences in this publication, are trademarks of other companies:

Adobe	Adobe Systems Incorporated
CompuServe	CompuServe, Incorporated and H&R Block, Incorporated
Intel	Intel Corporation
Microsoft	Microsoft Corporation
Microsoft C	Microsoft Corporation
NDIS	3Com Corporaton and Microsoft Corporation
PostScript	Adobe Systems Incorporated
UNIX	Unix System Laboratories, Inc.

Windows	Microsoft Corporation
X/Open	X/Open Company, Ltd.
3Com	3Com Corporation

About This Book

Networking Services is a software product that supports Advanced Program-to-Program Communication (APPC) in the Microsoft** Windows** 3.1 and Microsoft Windows for Workgroups** 3.1 environments. With Networking Services, communications applications on your workstation can “talk to” partner applications on other systems that support APPC.

This book explains how to:

- Write distributed applications for Networking Services
- Plan for conversations to be started by a remote computer
- Use the node operator application program interface

This book also explains the conditions under which you must make changes to existing programs if you are upgrading from Networking Services/DOS to IBM APPC Networking Services.

This book is also available in softcopy format for viewing with IBM* BookManager* READ or Library Readers*. IBM includes the Library Readers on all CD-ROMs that contain softcopy books. The Library Readers are limited editions of BookManager READ/2, READ/DOS, and READ for Windows that enable you to read and search softcopy books specifically enabled by IBM. The Library Reader for Windows is included in the IBM APPC Networking Services for Windows package.

Before you use BookManager to display this book, see the license agreement on page vii. For more information on BookManager READ for Windows refer to the *BookManager Library Reader for Windows: User's Guide*.

Who Should Read This Book

This book is intended for:

- Application programmers who write distributed applications for Networking Services using Common Programming Interface for Communications (CPI-C).
- Application programmers who write applications that use the node operator facility (NOF) application program interface to exercise program control of Networking Services functions.

The prerequisites for using this book are:

- Familiarity with the Windows graphical environment
- Familiarity with SNA networks in general, and your own network in particular.
- Familiarity with Common Programming Interface for Communications (CPI-C)
- Familiarity with workstations communications adapters and related concepts.

© Copyright IBM Corp. 1994

About This Book

Conventions Used in This Book

To help you locate and identify information easily, this book uses the following visual cues and standard text formats:

Type style	Used for
ALL CAPS	Directory names, file names, CPI-C call names, and keywords in the Networking Services network configuration file.
Bold	CPI-C conversation state names.
Italics	Titles of books, variables, and CPI-C call characteristics and variables.
Monospace	Names of entry fields, action bar items, choices from pull-down menus, and examples and messages displayed by Networking Services.
SMALL CAPS	CPI-C pseudonyms.

Terms Used in This Book

Terms used in this book include:

Term	Purpose
Type	Instructs you to type information in an entry field or file. You do not need to press Enter after typing this information.
Workstation	Refers to a programmable workstation.
KB	1024 bytes.
MB	1 048 576 bytes.

Part 1. Introduction

Chapter 1. Introduction 1-1

© Copyright IBM Corp. 1994

Chapter 1. Introduction

IBM APPC Networking Services for Windows is a software product that provides communications for Microsoft** Windows** programs that use Advanced Program-to-Program Communication (APPC). Networking Services provides support for the following application programming interfaces (APIs) that use APPC:

Common Programming Interface for Communications (CPI-C)
For more information about Networking Services's support for this API, see Part 2, "Writing Distributed Applications to the CPI-C API."

APPC Communications Router Application Program Interface
This API is also called the EHNAPPC API. For more information about this API, refer to the *AS/400 Client Access/400 for Windows 3.1 API and Technical Reference*.

In addition, Networking Services supports an API that can be used by DOS APPC programs. For information about this API, refer to "Router Service Verbs and Basic Conversation Verbs" in the *AS/400 Client Access/400 for Windows 3.1 API and Technical Reference*.

Also, Networking Services provides the **Node Operator Facility API** that application programs can use to request any of the following Networking Services functions:

- Starting and stopping Networking Services
- Connecting and disconnecting links
- Activating and deactivating partner logical units (LUs)
- Starting and stopping sessions
- Adding or deleting mode, side information, and transaction program definitions
- Changing the authorization of which incoming Attaches are to be added to the Attach list
- Removing an Attach from the Attach list
- Displaying current status

For more information about this API, see Chapter 6, "Using Networking Services Node Operator Application Program Interface" on page 6-1.

Note: If you plan to write applications using any of the API's listed above, install the **Toolkit** when you install Networking Services. All necessary files will be installed in the INCLUDE subdirectory of the directory where Networking Services is installed.

© Copyright IBM Corp. 1994

Introduction

Part 2. Writing Distributed Applications to the CPI-C API

Chapter 2. Using CPI-C Calls in Networking Services for Windows 2-1
Overview of CPI-C 2-1
Support of CPI-C Conformance Classes 2-2
Mandatory Conformance Classes 2-2
Optional Conformance Classes Supported 2-2
Optional Conformance Classes Not Supported 2-2
Non-Applicable CPI-C Terms and Concepts 2-3
CPI-C Calls Supported 2-4
Pseudonym Files 2-6
Examples of Using C 2-6
Linking With the CPI-C Import Library 2-7
Memory Considerations 2-7
Data Buffers 2-7
Stack Size 2-7
Defining Side Information 2-8
Usage Notes for Mode_Name and TP_Name 2-8
Mode_Name 2-8
Restrictions on Transaction Program Names 2-8
Multiple CMACCPs 2-8
How Dangling Conversations Are Deallocated 2-9
Diagnosing Errors 2-9
Log_Data 2-9
Identifying Product-Specific Errors 2-9
Deviations from the CPI Communications Architecture 2-11
Return_control Characteristic for Allocate (CMALLC) 2-11
CM_PROGRAM_PARAMETER_CHECK Return Code 2-11
Log Data Support 2-12

Chapter 3. Designing CPI-C Applications 3-1
Choosing an Application Model 3-1
The Client/Server Application Model 3-2
The Transaction Processing Application Model 3-3
The Distributed Application Model 3-3
The Real-Time Control Model 3-4
Combining CPI-C Calls to Perform Transactions 3-5
The Pipeline Transaction (One Way Bracket) 3-6
The Inquiry Transaction (Request and Reply) 3-7
The Credit-Check Transaction (Confirmed Delivery) 3-8
The File-Transfer Transaction (Batch Send) 3-10
The Database-Update Transaction (Conversational Reply) 3-11
Starting a Program 3-12
Choosing a Conversation Type 3-12
Implementing a Conversation 3-14
Allocating a Conversation to a Partner Application 3-15
Accepting a Conversation Started by a Partner Application 3-15

© Copyright IBM Corp. 1994

Deallocating a Conversation 3-15
Sending Data 3-15
Forcing Data to be Transmitted 3-15
Receiving Data 3-15
Granting Permission to Send 3-16
Requesting Permission to Send 3-16
Reporting an Error 3-16
Requesting Confirmation 3-17
Accepting or Rejecting a Confirmation Request 3-17
Extracting Conversation Information 3-17
Designing CPI-C Applications to Improve Performance 3-17

Chapter 4. Avoiding Deadlocks and Wait Conditions 4-1
Understanding Deadlock Situations 4-2
Understanding CPI-C Wait Conditions 4-2
Waiting for Data 4-2
Waiting for an Incoming Attach 4-3
Waiting for Pacing 4-3
Waiting for Confirmation 4-4
Waiting after a Deallocate or Send_Error Call 4-4
Identifying a CPI-C Application Deadlock 4-5
Stopping a Deadlocked CPI-C Program 4-5

Chapter 5. Managing Incoming Attaches 5-1
What Is the Attach Manager? 5-2
How Does an Incoming Conversation Start? 5-2
What Information Does the Attach Manager Need? 5-3
Processing Incoming Attaches 5-6
Adding Incoming Attaches to the Attach List 5-6
Calling the Program Launcher 5-7
Removing an Attach from the Attach List 5-7
Rejecting Incoming Attaches 5-8
Processing Accepts 5-8
Starting Programs and the Program Launcher 5-12
Starting Programs 5-12
Starting and Stopping the Program Launcher 5-12

Chapter 2. Using CPI-C Calls in Networking Services for Windows

This chapter contains information about the IBM APPC Networking Services for Windows implementation of and extensions to CPI Communications.

The topics in this chapter are:

Topic	Page
Overview of CPI-C	2-1
Support of CPI-C conformance classes	2-2
CPI-C calls supported	2-4
Pseudonym files	2-6
Linking with the CPI-C import library	2-7
Memory considerations	2-7
Defining side information	2-8
Usage Notes for Mode_Name and TP_Name	2-8
Multiple CMACCP calls	2-8
How dangling conversations are deallocated	2-9
Diagnosing errors	2-9
Deviations from the CPI-C architecture	2-11

Overview of CPI-C

CPI Communications (CPI-C) provides a cross-system-consistent programming interface for applications that require program-to-program communication. For a complete description of CPI-C, refer to the *CPI Communications Reference* or the *CPI Communications Specification*. For information on ordering these and other publications, see “Bibliography” on page X-1.

CPI-C is based on a conversational model of program-to-program communication. As you review the terms and concepts described in the referenced books, keep in mind that Networking Services for Windows supports only the APPC conversational model. It does not support the Open Systems Interconnection Distributed Transaction Processing (OSI TP) model. Therefore, the CPI-C **communication resource manager** (CRM) is known as a **logical unit** (LU) of type 6.2, and the **logical connection** between two CRMs is known as a **session**. For more information about which CPI-C terms and concepts do not apply to Networking Services for Windows, see “Non-Applicable CPI-C Terms and Concepts” on page 2-3.

© Copyright IBM Corp. 1994

Support of CPI-C Conformance Classes

Because CPI-C 2.0 provides a wide variety of communications functions, it has organized the functions into conformance classes in order to foster an orderly marketplace for CPI-C 2.0 implementations. Implementations must support certain required classes, and may also support other classes.

Mandatory Conformance Classes

In order to support a given class, an implementation must support all the CPI-C calls defined as part of that class. In addition, for some calls, certain parameters must also be supported. For more information about the definition of CPI-C 2.0 conformance classes, refer to the *CPI Communications Reference* or the *CPI Communications Specification*.

To support CPI-C 2.0, an implementation must support the conformance classes:

Conversations

Either the LU 6.2 or the OSI TP conformance class, or both

Networking Services for Windows supports the conversations class and the LU 6.2 class.

Optional Conformance Classes Supported

Networking Services for Windows supports the following optional conformance classes:

Data conversion routines

Full-duplex

Queue-level non-blocking

Callback function

Optional Conformance Classes Not Supported

Networking Services for Windows does not support the following optional conformance classes because it does not support all the calls that belong to each class. However, for some classes, Networking Services for Windows supports a subset of the required calls, as noted. The calls supported by Networking Services for Windows are shown in Table 2-1 on page 2-4.

Conversation-level non-blocking — Networking Services for Windows supports queue-level non-blocking instead

Directory

Distributed security

Expedited data

OSI TP

Recoverable transactions

Secondary information

Security — Networking Services for Windows supports all calls that provide security functions. However, the input parameter of CM_SECURITY_PROGRAM_STRONG on the call, Set_Conversation_Security_Type (CMSCST) is not supported.

Server — Networking Services for Windows supports the calls that provide the server function of accepting multiple incoming conversations, but not the calls that allow a program to register multiple TP names or to manage contexts

Unchained transactions

Non-Applicable CPI-C Terms and Concepts

Because Networking Services for Windows does not support all conformance classes, the following terms and concepts presented in the *CPI Communications Reference* and *CPI Communications Specification* do not apply:

Distributed directory or directory objects

Distributed security

Context management

Conversation-level non-blocking

Resource recovery interfaces

CPI-C Calls Supported

Networking Services for Windows supports the CPI-C calls shown in Table 2-1. The table also indicates the conformance class to which each call belongs.

The presence of a conformance class in Table 2-1 does not imply that Networking Services for Windows supports that class. In some cases, Networking Services for Windows supports some, but not all, CPI-C calls that belong to a conformance class. The conformance classes that are not supported by Networking Services for Windows are represented with parentheses.

For information about which conformance classes are supported by Networking Services for Windows, see "Support of CPI-C Conformance Classes" on page 2-2. For detailed descriptions of CPI-C calls, refer to the *CPI Communications Reference* or *CPI Communications Specification*.

Table 2-1 (Page 1 of 2). CPI-C Calls Supported by Networking Services for Windows

Pseudonym	Call Name	Conformance Class
Accept_Conversation	CMACCP	Conversations
Accept_Incoming	CMACCI	(Server)
Allocate	CMALLC	Conversations
Cancel_Conversation	CMCANC	Queue-level non-blocking
Confirm	CMCFM	Conversations
Confirmed	CMCFMD	Conversations
Convert_Incoming	CMCNVI	Data conversion routines
Convert_Outgoing	CMCNVO	Data conversion routines
Deallocate	CMDEAL	Conversations
Extract_Conversation_State	CMECS	Conversations
Extract_Conversation_Type	CMECT	Conversations
Extract_Maximum_Buffer_Size	CMEMBS	Conversations
Extract_Mode_Name	CMEMN	Conversations
Extract_Partner_LU_Name	CMEPLN	LU 6.2
Extract_Security_User_ID	CMESUI	(Security)
Extract_Send_Receive_Mode	CMESRM	Full-duplex
Extract_Sync_Level	CMESL	Conversations
Extract_TP_Name	CMETPN	(Server)
Flush	CMFLUS	Conversations
Initialize_Conversation	CMINIT	Conversations
Initialize_For_Incoming	CMINIC	(Server)
Prepare_To_Receive	CMPTR	Conversations
Receive	CMRCV	Conversations
Request_To_Send	CMRTS	Conversations
Send_Data	CMSEND	Conversations
Send_Error	CMSERR	Conversations
Set_Conversation_Security_Password	CMSCSP	(Security)
Set_Conversation_Security_Type	CMSCST	(Security)
Set_Conversation_Security_User_ID	CMSCSU	(Security)
Set_Conversation_Type	CMSCT	Conversations
Set_Deallocate_Type	CMSDT	Conversations
Set_Error_Direction	CMSED	LU 6.2
Set_Fill	CMSF	Conversations
Set_Log_Data	CMSLD	Conversations

Table 2-1 (Page 2 of 2). CPI-C Calls Supported by Networking Services for Windows

Pseudonym	Call Name	Conformance Class
Set_Mode_Name	CMSMN	Conversations
Set_Partner_LU_Name	CMSPLN	LU 6.2
Set_Prepare_To_Receive_Type	CMSPTR	Conversations
Set_Queue_Callback_Function	CMSQCF	Callback function
Set_Queue_Processing_Mode	CMSQPM	Queue-level non-blocking
Set_Receive_Type	CMSRT	Conversations
Set_Send_Receive_Mode	CMSSRM	Full-duplex
Set_Return_Control	CMSRC	Conversations
Set_Send_Type	CMSST	Conversations
Set_Sync_Level	CMSSL	Conversations
Set_TP_Name	CMSTPN	Conversations
Test_Request_To_Send_Received	CMTRTS	Conversations
Wait_For_Completion	CMWCMP	Queue-level non-blocking

Table 2-2 shows which calls are not supported by Networking Services for Windows and indicates which conformance class the call belongs to.

Table 2-2. CPI-C Calls Not Supported by Networking Services for Windows

Pseudonym	Call Name	Conformance Class
Deferred_Deallocate	CMETC	Unchained transactions
Extract_AE_Qualifier	CMEAEQ	OSI TP
Extract_AP_Title	CMEAPT	OSI TP
Extract_Application_Context_Name	CMEACN	OSI TP
Extract_Conversation_Context	CMECTX	Server
Extract_Initialization_Data	CMEID	OSI TP
Extract_Secondary_Information	CMESI	Secondary information
Include_Partner_In_Transaction	CMINCL	Unchained transactions
Receive_Expedited_Data	CMRCVX	Expedited data
Release_Local_TP_Name	CMRLTP	Server
Send_Expedited_Data	CMSNDX	Expedited data
Set _AE_Qualifier	CMSAEQ	OSI TP
Set _Allocate_Confirm	CMSAC	OSI TP
Set _AP_Title	CMSAPT	OSI TP
Set _Application_Context_Name	CMSACN	OSI TP
Set_Begin_Transaction	CMSBT	Unchained transactions
Set _Confirmation_Urgency	CMSCU	OSI TP
Set _Initialization_Data	CMSID	OSI TP
Set_Processing_Mode	CMSPM	Conversation-level non-blocking
Set_Transaction_Control	CMSTC	Unchained Transactions
Specify_Local_TP_Name	CMSLTP	Server
Wait_For_Conversation	CMWAIT	Conversation-level non-blocking

Pseudonym Files

Networking Services for Windows provides a pseudonym files to support the C language, CPIC.H.

This pseudonym file is located in the INCLUDE subdirectory on the disk where Networking Services for Windows is installed. You may want to add this subdirectory to your INCLUDE environment variable.

These files contain constant declarations and data types for each supplied and returned parameter in CPI-C calls. To use another language or a non-standard implementation of a supported language, you will have to write a pseudonym file. If you are using another language, use the provided file as the model to build the pseudonym file for your language. Other pseudonym files are listed in the *CPI Communications Reference*.

When writing pseudonym files, note that the Networking Services for Windows implementation of CPI-C follows the Microsoft Pascal calling conventions. In the library files, NSDW.LIB and NSDW.DLL, the symbol names corresponding to the CPI-C calls are normalized to uppercase. This is important when using a language or compiler that is case-sensitive or when linking with the "/NOIGNORECASE" option.

CPI-C calls do not return a value; they are procedures rather than functions. The CPI-C API is invoked by far calls only. Parameters are passed by reference using far pointers pushed onto the stack in the order in which they lexically appear in the pseudonym files (*conversation_ID* first). The called CPI-C procedure removes its received parameters from the stack before returning to the caller.

Examples of Using C

One of the installation options for Networking Services for Windows is whether you want the sample programs. If you select this option, the installation program installs the sample programs and source code in the SAMPLES subdirectory. These samples are a guide to the details of compiling and linking real applications.

CPI-C Function Calls in C

The following example illustrates the Initialize_Conversation call in the C language:

```
unsigned char conversation_id[8];       / your conversation ID          /
unsigned char sym_dest_name[8];         / symbolic destination name    /
CM_INT32 cm_retcode;                    / return code                  /

cminit(conversation_id, / Address of returned conversation ID      /
       sym_dest_name,   / Address of supplied destination address /
   &cm_retcode);        / Address of returned return code         /
```

Using the Pseudonym Files in C Language Programs

The pseudonym files contain function prototypes for each of the CPI-C library calls. These function prototypes help the compiler check for agreement of the number and type of each parameter.

The C pseudonym file, CMC.H., contains the CPI-C constants and function prototypes.

Using Other Languages

Determine how to make calls using the Pascal calling convention in your language. The calling convention refers to the order in which procedure parameters are pushed on the stack. When multiple parameters are passed on a call, they can be pushed onto the stack in two ways:

C Push the last parameter first, then the next-to-last, and so on.

This calling convention is not supported in Networking Services for Windows.

Pascal Push the first parameter first, then the second, and so on. Your compiler would use a sequence like the following to make a CPI-C call.

```
PUSH the 32-bit address of the first parameter
PUSH the 32-bit address of the second parameter
:
PUSH the 32-bit address of the last parameter
CALL the 32-bit address of the specific CPI-C call
```

Linking With the CPI-C Import Library

Link your application with the CPI-C import library, NSDW.LIB, shipped with Networking Services for Windows. This library is located in the Networking Services for Windows LIB directory.

Memory Considerations

Data Buffers

For calls that require data buffers, such as Receive or Send_Data, your program can either declare the data buffer as a variable or allocate it using a library function (for example, the C `malloc()` function).

If your program specifies a buffer length greater than the actual buffer size, Networking Services for Windows may not detect this error. As a result, data lying beyond the end of the buffer may be overwritten.

Beware when allocating memory using a library function that can allocate more than 64KB (for example, the C `halloc()` function). No data buffer passed to Networking Services for Windows can be larger than 64KB.

Stack Size

Networking Services for Windows calls execute on the calling application's stack. You are responsible for ensuring that your application's stack is sufficiently large.

Defining Side Information

In Networking Services for Windows, the Initialize_Conversation call reads side information in order to find the partner LU, transaction program, and mode names associated with the specified *sym_dest_name*. For more information on creating side information for Networking Services for Windows, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

Usage Notes for Mode_Name and TP_Name

Mode_Name

Although Networking Services for Windows allows you to set the *mode_name* characteristic to CPSVCMG or SNASVCMG using the Set_Mode_Name (CMSMN) call, it rejects an Allocate (CMALLC) call for a conversation with either of these mode names and returns a CM_PARAMETER_ERROR return code.

Specification of the blank mode for a conversation is handled in different ways depending on where the mode is set. When the mode is specified in the side information table, the string 'BLANK' must be used. When the mode is set by means of a Set_Mode_Name (CMSMN) call, an empty string or a string containing only ASCII blanks must be specified. If you use the Extract_Mode_Name (CMEMN) call on a session using the blank mode, Networking Services for Windows returns a zero-length string.

Restrictions on Transaction Program Names

Networking Services for Windows translates transaction program names from ASCII to EBCDIC. Therefore, if a program sets the *TP_name* characteristic using a double-byte name and then calls Allocate (CMALLC), the partner LU rejects the allocation request because of an incorrect transaction program name. Transaction program names can be entered in the side information table file using a hexadecimal format specification to allow any valid SNA transaction program name. For more information about valid formats for transaction program names, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

Multiple CMACCPs

Networking Services for Windows allows a program to execute multiple Accept_Conversation (CMACCP) calls. A CMACCP is allowed at any time that an Initialize_Conversation (CMINIT) would be allowed. However, a CMACCI is allowed only after an Initialize_For_Incoming call has been processed. See Chapter 5, “Managing Incoming Attaches” for more information about how Networking Services for Windows processes CMACCPs and CMACCIs.

How Dangling Conversations Are Deallocated

When an application ends without deallocating its active conversations, those conversations are *dangling*. Networking Services for Windows automatically deallocates dangling conversations, and also deactivates the sessions carrying those conversations related to that application.

Dangling conversations are deallocated when the use count of the shared NSDW.DLL goes to zero. Note that the Program Launcher uses the DLL; if you want Networking Services for Windows to automatically deallocate dangling conversations, stop the Program Launcher.

Diagnosing Errors

When a Networking Services for Windows call returns an error return code, Networking Services for Windows places a message in the message log. If tracing is enabled, Networking Services for Windows also places a message in the trace log.

Log_Data

A successful call to Set_Log_Data (CMSLD) sets the *log_data* and *log_data_length* characteristics. However, log data is not sent to the conversation partner; it is only entered into the trace log when tracing is enabled.

When tracing is enabled, Networking Services for Windows places log data in the trace log when successfully entering:

Set_Log_Data (CMSLD)

Deallocate (CMDEAL).

Networking Services for Windows also places log data in the trace log when successfully leaving Set_Log_Data (CMSLD). However, at these times, Networking Services for Windows does not place a message in the message log. Also, Networking Services for Windows does not send *log_data* to the remote program.

Identifying Product-Specific Errors

To assist in identifying product-specific errors, enable CPI-C error tracing.

CPI-C allows a return code of CM_PRODUCT_SPECIFIC_ERROR when there is no corresponding CPI Communications return code to which the error could be mapped appropriately.

Networking Services for Windows returns the CM_PRODUCT_SPECIFIC_ERROR return code when:

Networking Services for Windows has not been started.

The underlying APPC implementation fails. In this case, Networking Services for Windows writes a message to the Networking Services for Windows message log. The message states that an APPC error occurred and gives the primary and secondary APPC return codes for this error.

Some internal error occurs, such as a call failing in its attempt to allocate storage internally for its own use. In this case, Networking Services for Windows writes a message to the message log, which indicates the nature of the failure.

Deviations from the CPI Communications Architecture

Deviations from the CPI Communications Architecture are found in the following areas:

Deviation	Page
Return_control characteristic for Allocate	2-11
Parameter checking	2-11
Log data support.	2-12

Return_control Characteristic for Allocate (CMALLC)

If a program issues Allocate (CMALLC) with return_control set to CM_WHEN_SESSION_ALLOCATED while no sessions are available and the session limit has been reached, Networking Services for Windows will not wait for a session to become available. Instead, the CM_ALLOCATE_FAILURE_RETRY return code will be returned.

CM_PROGRAM_PARAMETER_CHECK Return Code

A Networking Services for Windows call returns CM_PROGRAM_PARAMETER_CHECK if it detects that one or more of the parameters supplied in the call are incorrect. Networking Services for Windows also places a message in the message log. If CM_PROGRAM_PARAMETER_CHECK is not returned, it does not mean that the actual parameters are valid.

Networking Services for Windows returns CM_PROGRAM_PARAMETER_CHECK if it determines that one or more of the actual parameters is a protected mode address that is not valid. Examples include selector values that specify a *conversation_ID* that cannot be read, or a return code that cannot be written.

Log Data Support

Networking Services for Windows neither sends nor receives *log_data*. If any *log_data* is received, Networking Services for Windows discards it.

Chapter 3. Designing CPI-C Applications

This chapter introduces guidelines for designing applications that make CPI-C calls.

Topics covered in this chapter include:

Guidelines for Designing CPI-C Applications	Page
Choosing an application model	3-1
Combining CPI-C calls to perform transactions	3-5
Starting a program	3-12
Choosing a conversation type	3-12
Implementing a conversation	3-14
Improving the performance of CPI-C applications	3-17

Choosing an Application Model

An application model describes the relationships between a transaction program and its partners. This chapter discusses:

Application Models	Page
Client/server	3-2
Transaction processing	3-3
Distributed application	3-3

These models provide a framework to help you answer design questions, such as:

When and how is the application started?

When is a conversation allocated and deallocated?

When is the partner application to be given permission to send?

When is an error reported, how is the error identified, and what error recovery is performed?

When is confirmation requested? When is it safe to accept confirmation?

When is data sent? How is the data formatted?

Should an application force its data to be sent to its partner?

When is data received? What data checking must be performed before the data is used?

© Copyright IBM Corp. 1994

Choosing an Application Model

The Client/Server Application Model

The client/server application model centralizes your development effort into a few programs that other programs use to access a distributed resource. A distributed resource could be any of the following:

A file

A database

A printer

A library of subroutines

A workstation operator.

A client application provides all authorized programs in its node with access to some distributed resource. To access the distributed resource, the client application exchanges information with a server application that manages the distributed resource.

When an application requires access to the distributed resource, it makes a request of the client program in its node. The client program performs the necessary functions, and then replies to the request.

After initialization, the client waits for a request from a program in its node. When a request arrives, the client may verify the authority of the program to make the request. The client then attempts to perform the requested function before waiting for the next request:

If the function can be performed locally, the client performs the function, and then replies to the request.

If the function requires remote processing, the client allocates a conversation to a server, exchanges information with the server, deallocates the conversation, and then replies to the request.

After initialization, the server waits for any client to allocate a conversation. When a new conversation arrives, the server accepts the conversation. The server can verify the authority of the client or the client's user to make the indicated request. The server then exchanges information with the client. If the server cannot perform the function by itself, it can receive help from other servers by sending a request to a client in its node or by allocating a conversation directly to another server. Finally, the server deallocates the client's conversation and waits for the next conversation.

Characteristics of the client/server application include the following:

The communications code is centralized in the clients and servers; the code is not added to each program that needs access to a distributed resource. When many programs need access to the same distributed resource, this technique significantly reduces the development effort.

These conversations tend to be short in duration. Typically, a conversation between a client and a server lasts just long enough to satisfy the user's request. When the conversation is deallocated, other conversations can use network resources, such as the session and buffers. By sharing network resources among

many conversations from the same or different clients, you reduce network usage and management costs.

The Transaction Processing Application Model

The transaction processing application model divides the application into a collection of transactions. A single request initiates each transaction, and a single program processes the transaction. While processing a transaction, a program can start transactions with other programs.

When a user or an event initiates a transaction, the transaction program formats a record containing a single request, and allocates a conversation to the program written to process that request. The target program performs the request, exchanging data with the first program as necessary. When the program finishes processing the request, it deallocates the conversation.

Characteristics of the transaction processing application model include the following:

A single program that handles many different kinds of requests is more difficult and expensive to develop and maintain than a group of small, simple programs, each of which processes a single kind of request. Typically, a different target program is used for each different type of transaction; TP names distinguish multiple targets. Thus, with the transaction processing model, changes and enhancements to programs are localized.

This model is well-suited for dealing with IBM transaction processing systems, such as:

- Transaction Processing Facility (TPF)
- Customer Information Control System (CICS)*
- Information Management System (IMS).

These systems manage large libraries of programs. They manage the necessary resources to automatically load and run each program when its transaction is received. Each system can run multiple instances of the same program to process identical requests simultaneously.

The Distributed Application Model

This model distributes an application between two or more programs. Each program exchanges data with its peers to distribute the tasks of the application.

Some reasons for distributing an application are:

- To distribute the workload to nodes that have adequate capacity.
- To distribute application functions to nodes that are capable of performing them.
- To perform a certain application function on a node that provides a required resource, such as a printer, a database, a user, or a program.

You can use any number of programs to implement your distributed application, although this application model also works well with a single pair of programs. The

maximum number of programs is determined by the capacity of the network and the nodes where the programs run.

Characteristics of the distributed model include the following:

- The program can communicate with other programs transparently. The program uses exactly the same procedures regardless of the location of the other program.
- The partner program can be automatically started when the conversation is requested.
- The programs can synchronize using the features provided by CPI-C.
- Conversations last until the programs are no longer active.
- Some application designs require a program to accept multiple incoming conversations. Networking Services allows a program to accept multiple incoming conversations. Other CPI-C implementations may restrict the number of conversations a program can accept. For more information about how Networking Services accepts multiple incoming conversations, see “Processing Accepts” on page 5-8.

Typically, the conversations in the distributed application model are long in duration. A long-lived conversation can tie up system and network resources, even when it is idle for long periods. If other conversations are needed while your conversations are active, increase the number of allowed sessions on your system and other systems in the network. However, increasing the number of sessions can increase the cost of using and managing the network.

If your application permits it, you may be able to share resources by periodically deallocating conversations and allocating them again when they are needed. The programs receiving the incoming conversations may need to accept multiple conversations without ending and restarting the programs each time. The systems at which those programs are located must support this capability.

The Real-Time Control Model

A real-time control application simultaneously sends commands and receives feedback messages that are fed into a real-time function. The output from the function is used to generate new commands that must be sent promptly to avert disastrous consequences. Typically, there is no relationship between the number of feedback messages and the number of command messages.

Such an application should use full-duplex conversations, because both feedback messages and command messages may need to be sent at the same time. As far as CPI-C is concerned, the streams of records being exchanged are completely independent. The application must contain logic to understand the relationship between the records.

Combining CPI-C Calls to Perform Transactions

Five types of transactions form the basis from which most communicating applications can be constructed. This section describes how to use CPI-C calls to perform these transactions. They are introduced in order of increasing complexity.

Transaction Types	Page
Pipeline transaction	3-6
Inquiry transaction	3-7
Credit-check transaction	3-8
File-transfer transaction	3-10
Database-update transaction	3-11

Notes about the examples for each type of transaction:

The examples include the underlying SNA line flows. This information was added for completeness. No knowledge of line flows is necessary to write CPI-C applications.

Three abbreviations are used in this section:

rc= indicates the CPI-C return code

dr= indicates the returned data_received value

sr= indicates the returned status_received value

The examples include only the parameters meaningful within the context of the discussion.

Combining CPI-C Calls for Transactions

The Pipeline Transaction (One Way Bracket)

A pipeline transaction sends a message to which no reply is expected. This transaction is also known as the “one way bracket” because the SNA line flow bracket begins and ends without generating any return flow.

In this transaction, the client sends a chunk of data and then leaves without receiving an acknowledgment. The client doesn't know whether the server received or completely processed the incoming record. If the application does include a reply, the programs must correlate their requests and replies across conversations.

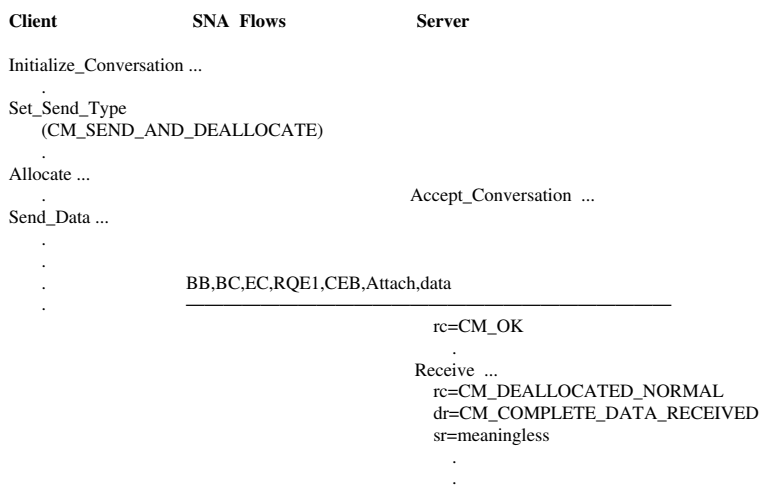

Figure 3-1. Pipeline (One Way Bracket)

This is the classic “short conversation.” For example, APPN* network nodes use this transaction type to transmit topology updates and to implement resource searches. This transaction is used when the arrival rate of short transactions is too large to allow the use of a session for every active transaction. This can occur when a node contains insufficient storage or when a very large number of transactions must be run in parallel.

In these situations, write an application that uses a full-duplex *send_receive_mode*. Alternatively, you can write the application to use two half-duplex conversations: one to send data and one to receive data. The application builds a correlator into each data record to identify the unit of work to which it belongs. The partner returns the correlator in related data records so that the application can correlate the data to the correct unit of work.

The Inquiry Transaction (Request and Reply)

The inquiry transaction requests information from a server program. This transaction is also known as "request and reply" because the entire conversation consists of one request and one reply. In the following figure, a client sends a request to a server that obtains the answer and replies.

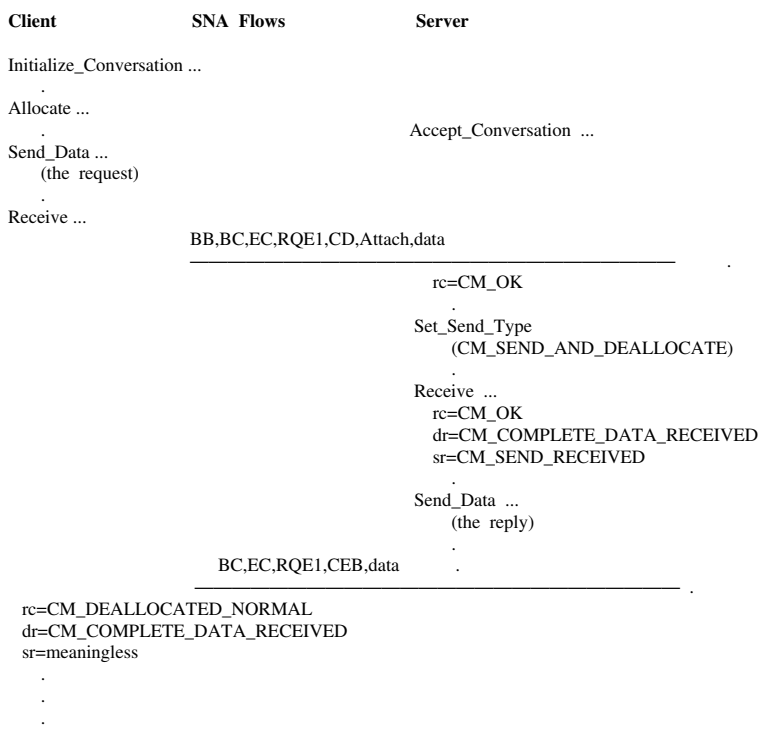

Figure 3-2. Inquiry (Request and Reply)

Banks and retailers, for example, frequently use inquiry transactions. These types of transactions are appropriate for the following activities:

Remote Procedure Calls
A client program sends a subroutine call request to a server program. The server program performs the subroutine and returns the result.

Database Applications
A client program sends a set of keys to the database server, which then returns the data record selected.

Status Queries
A client program wants to know, "Are you there, and what is your status?"

The Credit-Check Transaction (Confirmed Delivery)

The credit-check transaction is used when a client program must request permission to perform a specified function and the server will probably grant permission. This transaction is also used when a client program must send information to the server and needs a guarantee that the information has been successfully received and processed. For this reason, the transaction is also called the “confirmed delivery” transaction.

In the credit-check transaction, unlike the inquiry transaction, the server returns no data. Instead, the server returns an acknowledgment that it received and processed the data sent by the client program.

A program designed to do credit checks illustrates the usefulness of this type of transaction. The client program:

1. Builds a credit-check record, which includes an authorization code that is returned to the salesperson if the purchase is authorized
2. Sends the record to the credit-check server
3. Requests a confirmation.

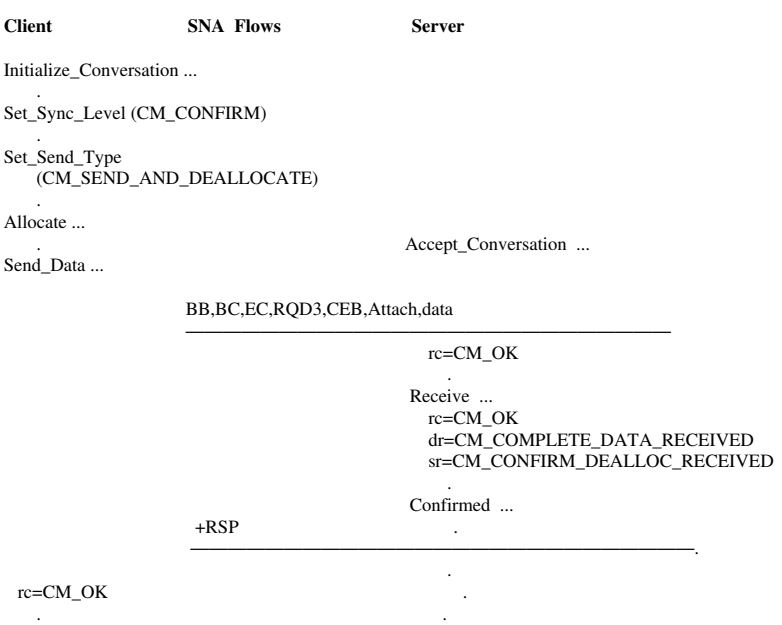

Figure 3-3. Credit Check (Confirmed Delivery)

If the credit check is successful, no reply is necessary; the server accepts the information. Only a confirmation[1] returns to the client. Because a reply is not formatted, sent, or received, the overhead and turnaround time is minimal.

1 In LU 6.2, a confirmation is implemented with the shortest possible message, an SNA positive response.

If the credit check is not successful, the server rejects the confirmation by reporting an error (using a Send_Error call) and then sends an error record that explains why the credit check failed.

With this technique, the server does not have to send a reply if it processes the transaction successfully. If the transaction is successful at least 80 percent of the time, the credit-check transaction produces better performance than the inquiry transaction.

The credit-check transaction is appropriate for the following activities:

Permission
A client program needs permission from a controlling server before continuing.

Polls
A client program wants to know, “Are you there?” The reply implies, “Yes, I am here.”

Alerts
A message reports an error condition that may require human intervention to fix. For example, SNA Management Services uses this type of transaction to deliver an Alert from a node to its focal point.

Combining CPI-C Calls for Transactions

The File-Transfer Transaction (Batch Send)

The file-transfer transaction sends relatively large amounts of data from one location to another or moves several related data records. It is called the “batch send” transaction because it sends a batch of records at the same time.

This transaction, an extended version of the credit-check transaction, shows the buffering capabilities of CPI-C. Using CPI-C, the client and server send and receive records without worrying about record boundaries or record sizes.

In Figure 3-4, the client uses one or more Send_Data calls to transfer the contents of a file. The server receives data with a loop of Receive calls until it has received all the data.

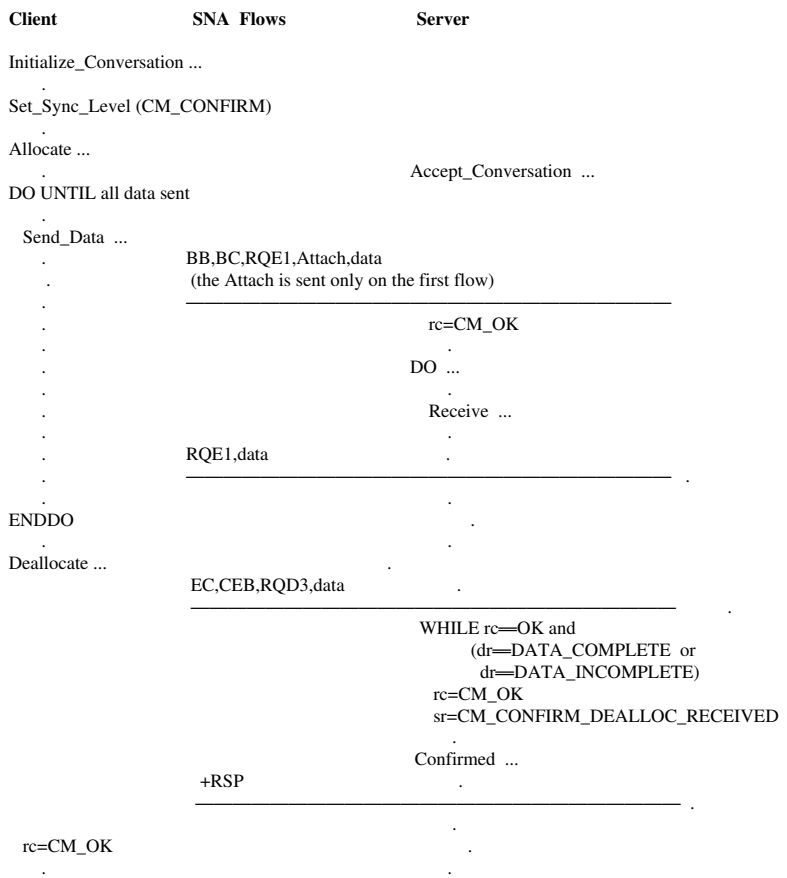

Figure 3-4. File Transfer (Batch Send)

The Database-Update Transaction (Conversational Reply)

The database-update transaction updates information managed by a partner program. In database applications, the client program requests information from the database server, updates the information, and returns the update to the server to be set into the database. This transaction is called a “conversational reply” because the reply solicits an additional reply from the originator.

This transaction is essentially an inquiry transaction followed by a credit-check transaction in the same conversation. Use this transaction when a client requests information from a server and the server, in turn, requests information from the client. As illustrated in Figure 3-5, the server appends the new request to the initial reply and returns the result in another record.

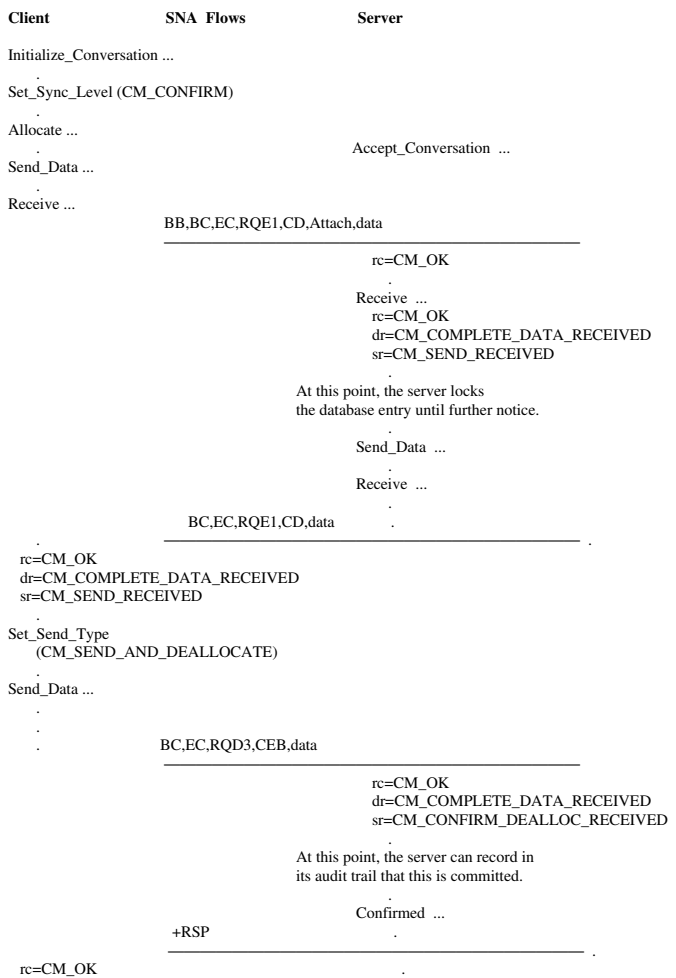

Figure 3-5. Database Update (Conversational Reply)

Starting a Program

To initiate and accept conversations, you must first start the application that contains the CPI-C calls. The following guidelines can help you decide how and when to start your CPI-C application.

If your program runs on an attended workstation, the user can start the program or you can allow the program to be started automatically. To start your program automatically, let the Networking Services Program Launcher start your program when it receives an incoming Attach.

See Chapter 5, “Managing Incoming Attaches” for additional information about the Program Launcher.

Choosing a Conversation Type

The two types of conversations are *mapped* and *basic*. While most CPI-C features are the same for both type of conversations, the way programs deal with data in the data buffers differs slightly.

A mapped conversation data buffer contains one data record. Applications use mapped conversations to send or receive one data record at a time. For example, file transfer applications often use mapped conversations. To send 25 000 bytes at a time, these applications move through the file, reading 25 000 bytes at a time into the data buffer.

A basic conversation data buffer may contain any number of logical records in a buffer. For example, a basic conversation data buffer may contain:

- One record
- The beginning of a record
- The middle of a record
- The end of a record
- One or more records and the beginning of another record
- The end of a record followed by one or more records.

When your application fills the data buffer, it must put a 2-byte logical length field called the *LL field* in front of each record. The number of records placed in a data buffer varies, depending on your transaction and its response-time needs.

In basic conversations, the LL field determines the logical record length in bytes, which includes the 2-byte length of the LL field itself.

- An LL field of X'0002' (that is, the NULL record) is allowed.
- An LL field less than X'0002' is not allowed.

After finding a valid LL field, the LU ignores the contents of the record, because the LU does not look for the next LL field until the end of the record. Therefore, the application can place anything it wants in the record, including LL fields that precede variable-length subfields.

The size of a single logical record is limited to 32 767 bytes. The high-order bit on the LL field is reserved for your application and is ignored by the LU. For structures larger than 32 767 bytes, the application can use the high-order bit to chain multiple logical records together. When the application turns this bit on, all subsequent records are chained together until the application turns the bit off.

You can put any number of logical records in your application's basic conversation buffer. In addition, records can span your application's buffer when the logical record is larger than the buffer size. In this case, you send or receive a buffer's worth at a time. Make sure that you send all the bytes indicated by the LL field.

Note: CPI-C basic conversation buffers are limited to 32KB.

Records and data buffers are the only parameters controlled by the application. The process that CPI-C uses to transfer data between two computers is independent of your applications, their record sizes, and data buffer sizes.

The mapped conversation API is intended for general applications. The basic conversation API is intended for more low-level programs.

Some guidelines for selecting a conversation type are:

If your application communicates with an existing application, choose the conversation type expected by the partner application.

If your application must process logical records that include length fields, choose basic conversations.

Otherwise, always choose mapped conversations.

Implementing a Conversation

Your program either starts a conversation or accepts one from a program that started it. After starting or accepting a conversation, the programs communicate to carry out the mission of the application. This section provides guidelines on the following CPI-C functions and how to use them:

CPI-C Functions	Page
Allocating a conversation	3-15
Accepting a conversation	3-15
Deallocating a conversation	3-15
Sending data	3-15
Forcing data to be transmitted	3-15
Receiving data	3-15
Granting permission to send	3-16
Requesting permission to send	3-16
Reporting an error	3-16
Requesting confirmation	3-17
Accepting or rejecting a confirmation request	3-17
Extracting conversation information	3-17

Allocating a Conversation to a Partner Application

If your program is a client or requester program or a cascading server (a server program that in turn requests conversations with other servers), your program must allocate a conversation with a server program. If your program initiates a transaction, it must allocate conversations to the subordinate programs involved in the transaction. These subordinate programs can allocate conversations to yet other subordinate programs.

When your program allocates a conversation, your workstation sends at Attach. The system receiving the incoming Attach starts a new partner application instance. The system can load and start a program defined for that application instance, or it can start a new application instance for a program that is already running.

Accepting a Conversation Started by a Partner Application

If your program is a server program, code it to accept each conversation as it arrives. See Chapter 5, “Managing Incoming Attaches” for additional information about the Program Launcher and the CMACCP and CMACCI call descriptions.

If your program is a subordinate application for a transaction, the incoming conversation starts the program. After the program is started, it accepts the incoming conversation and proceeds with the conversation.

Deallocating a Conversation

When the conversation between the programs is finished, either program can deallocate the conversation. The LU informs the other program that the conversation has been deallocated and releases any network resources. When the resources are released, the LU uses them to satisfy queued or future requests from any program.

Sending Data

After issuing a Send_Data call, do not expect the local LU to transmit the data immediately. Because the LU tries to reduce network overhead, it places your data into an internal buffer, and then returns control to your program. The local LU transmits the contents of the internal buffer when your program sends more data than fits in the buffer or when your program forces the data to be transmitted.

Forcing Data to be Transmitted

Your program can force the local LU to transmit data stored in internal buffers. For example, a program sends several records, then discovers there will be a short delay before the next record will be ready to send. During the delay, the other program can process the data records that arrived so far (parallel processing). In this case, your program should issue a call to force the local LU to transmit its internal buffer.

Receiving Data

Your program can ask the local LU to wait for all the data to arrive or to return the data that has arrived so far. If your program's data buffer is large enough to receive all the data, the local LU returns the data and informs your program how much data it returned. If your program's data buffer is not large enough, however, the local LU fills

the buffer and keeps the rest of the data, so that it will be received with the next request.

Receiving data from a partner program is similar to reading sequential records from a disk file. Your program does not read a specific number of records; rather, it reads records until the access method returns an end-of-file indication. With CPI-C, the LUs enforce this same logic. Unless it encounters an error, your program is coded to continue receiving data records until the local LU informs it that all records have been sent.

Granting Permission to Send

CPI-C offers two modes for sending and receiving data:

Two-way alternate data transfer, also known as half-duplex (flip-flop) data transfer.

Two-way simultaneous data transfer, also known as full-duplex data transfer.

With a full-duplex conversation, both conversation partners have permission to send at the same time. With a half-duplex conversation, only one program has permission to send data, at any point in time. Using half-duplex protocols can reduce the complexity of your programs by eliminating the need to account for the sequence and time of the data sent by the two programs.

When a half-duplex conversation is allocated, the program allocating the conversation has permission to send. When a program with permission to send finishes sending data and wants to receive the reply, it grants the other program permission to send. Thus, permission to send alternates between the programs.

Requesting Permission to Send

If your program uses a half-duplex conversation, has successfully received data and wants to send some data to the other program, your program must request permission to send.

CPI-C treats permission to send on a half-duplex conversation as an agreement between a pair of programs. The local LU does not force the other program to grant your program permission to send. Instead, it informs the other program that your program requested permission to send. The other program can either grant permission to send or ignore the request. The choice depends upon how you design the programs.

Programs using full-duplex conversations do not need to request permission to send.

Reporting an Error

If your program receives an incorrect data record or cannot perform a requested service, it must inform the other program. In addition, if an error prevents your program from sending a data record, your program must report the situation to the other program. Your program does this by issuing a Send_Error (CMSERR) call.

When your program reports an error, your LU automatically gives your program permission to send. As a result, your program can send a data record indicating what type of error was detected and begin appropriate recovery processing.

Note: When both programs using a half-duplex conversation report an error at the same time, the program that was receiving data has permission to send. The other program's error notification is ignored. In a full-duplex conversation both programs always have permission to send; the error notifications of both programs may be useful to the application.

Requesting Confirmation

Confirmation is used only on half-duplex conversations.

At times, your program may not continue until it knows that the other program received and successfully processed all the data that was sent. For example, consider a program that moves data records from one database to another. The program cannot delete the records from the first database until it knows that the other program has successfully entered the records into the second database.

When your program requests confirmation (using, for example, the Confirm call), the local LU first transmits all the data records it has buffered for that conversation. After the other program receives all the data records, your program requests confirmation. The other program either transmits the confirmation (using the Confirmed call) if the data records were processed successfully or rejects the confirmation request (using the Send_Error call).

Accepting or Rejecting a Confirmation Request

Confirmation is used only on half-duplex conversations.

Another program may request a confirmation from your program. If your LU informs your program of the confirmation request, your program must complete any unfinished processing and determine its response to the request. If the processing is successful, the confirmation is transmitted by issuing the Confirmed call. Otherwise, the confirmation request is rejected by issuing the Send_Error call.

Extracting Conversation Information

Your program may need to extract conversation information for debugging, problem determination aids, audit trails, high-level security checks, or other purposes. CPI-C enables your program to extract the conversation state, the conversation type, the mode name, the partner LU name, the sync_level, the security_user_id, the TP_name, the maximum_buffer_size, and the send_receive_mode of the conversation.

Designing CPI-C Applications to Improve Performance

The best way to improve the CPI-C performance of your applications is to combine control indicators with the sending and receiving of data. This reduces the number of CPI-C calls to the CPI-C interface. Specific ways to minimize the number of calls to CPI-C follow.

Designing CPI-C Applications

Use full-duplex conversations when appropriate to your application.

By using full-duplex, your application can eliminate some calls that are necessary to request and grant permission to send.

Use non-blocking calls.

Non-blocking calls allow your program to perform other work while waiting for operations to complete.

When using half-duplex conversations, minimize changing conversation states between Send and Receive.

Send as much data as possible before receiving or requesting a confirmation.

Use the Set_Send_Type call to combine operations on Send_Data.

An application can combine other operations with each Send_Data, such as Flush, Confirm, and Deallocate. Such combinations can save extra calls to CPI-C.

Avoid overusing the Flush call.

Flush forces CPI-C to send the contents of its current RU, even though the RU may not be full. Unless the partner needs the data immediately, let your LU determine when to send the data. For example, do not use Flush when an application issues multiple Send_Data calls. CPI-C normally fills an RU before sending it. If the maximum frame size is 2KB and each Send_Data is 1KB, flushing the RUs doubles the number of transmissions and uses only half of the frame buffers. The size of the RU buffer is defined in the mode definition. For more information, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

In contrast, if your application issues a Send_Data call and then issues no CPI-C calls immediately, the application should issue a Flush call to clear its buffers so that the remote program can use the data.

When using half-duplex conversations, minimize use of Confirm call.

Confirm forces an application to wait for an explicit acknowledgment from the partner. Avoid using the confirm functions unless positive acknowledgment is required. Rather than requesting confirmation after each Send, partner applications should issue a Send_Error only when they detect a failure in the data being transmitted.

Minimize use of Send_Error call.

Send_Error notifies the remote program that an error was detected. Do not use Send_Error calls as part of the mainline Send-Receive path of your application. Instead, use this call only when either partner detects a real error situation.

When using half-duplex conversations, minimize use of Request_To_Send call.

Code your pair of applications to allow the partner application to always be in send state if it needs to send information. Using the call issued, the return codes and the what_received indicators, the pair of applications know the state of the conversation at all times.

Minimize use of Extract calls.

Your applications should know most of this information. If your applications include these calls, the code should reserve the returned parameters in local program variables for later use.

Because these calls can be helpful on error paths, do not omit them entirely from your programs. For example, if a return code indicates that an application issued a CPI-C call in the wrong state, you can use the Extract_Conversation_State call to find the current state of the conversation.

Send many complete logical records with one call.

If you use basic conversations, send as much data as possible on each Send_Data. If the data is available and less than 32KB, your application can send multiple LLs with a single Send_Data call. However, remember that larger amounts of data mean that more space is needed for buffers within your application. Avoid splitting LL fields across calls. See “Choosing a Conversation Type” on page 3-12 for additional information about LL fields.

If you use mapped conversations, send as much data as possible on each Send_Data (32 767 bytes). If your application writes to disk files, consider making the data buffer size evenly divisible by the disk sector size.

Receive as much data as possible on each Receive call.

Allocate a large memory block for incoming data, and receive as much data as possible on each Receive call.

When using half-duplex conversations, deallocate only one side of a conversation.

Either partner may deallocate a half-duplex conversation, thereby ending the conversation for both partners. If the local program receives a deallocation return code it does not need to issue a Deallocate call. The deallocation return codes are:

```
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_DEALLOCATED_NORMAL
```

Avoid using any of the ABEND types on the Deallocate call.

The deallocate_type(Flush) is faster than the other types on the Deallocate call.

Use basic conversation calls only when absolutely necessary.

- Mapped conversation calls are easier to use than basic conversation calls. Mapped conversation calls require slightly more overhead inside CPI-C.
- With basic conversation calls you can send multiple logical records on each call to reduce the calls to CPI-C when sending groups of small records.

Keep conversations active when calling CPI-C frequently.

CPI-C sets aside internal buffers and control blocks each time a conversation is allocated and frees these each time a conversation is deallocated. When possible,

keep a running conversation active as long as it is needed, and then deallocate it when your application finishes with it.

However, if long pauses develop between an application's calls to CPI-C, it is better to deallocate the conversation and then to allocate a conversation again later. Be sure to think about the following:

- Consider the time required for the partner to process an incoming Attach. Attach processing can take variable amounts of time depending on the remote computer's characteristics and the way in which it is operated. The application designer should take this into account when choosing how often to create a conversation. On the other hand, there is also an overhead associated with keeping a conversation around when it is not doing any useful work. The proper choice is a trade-off that must be determined for each application environment.
- Consider the number of sessions required.

  If your application maintains a conversation for an indefinite period, other applications may require their own sessions. If other Networking Services applications on your workstation communicate with applications serviced by your partner LU, they probably wait for a session.
- Consider processing delays.

  If the delay is long, other applications can use the network resources. If the delay is very short, deallocating the conversation can cause excess system processing. If your application will not use the conversation for an extended time, you may want to deallocate it and allocate a new conversation later when it is needed. This action frees up network resources while they are unused, but costs extra processing and flows to end and restart the conversation.
- Consider the server resource utilization when you use CPI-C in a server-requester relationship.

  By deallocating the conversation, your partner can allocate the program space, buffers, and control blocks to other requesters.
- Consider the cost of the link.

  Design sessions and conversations to be as short as possible if you use long-distance telephone calls over asynchronous or switched SDLC links.

Chapter 4. Avoiding Deadlocks and Wait Conditions

This chapter explains how to avoid deadlocks and wait conditions when you design your CPI-C applications.

Deadlocks most often occur when one CPI-C call blocks the completion of another call. You can minimize the chance of a deadlock by using the queue-level non-blocking feature. For more information about queue-level non-blocking, refer to the *CPI Communications Specification* or the *CPI Communications Reference*.

Wait conditions occur when an application waits indefinitely for its partner to take some action.

Because these conditions lead to situations that are unpredictable and difficult to reproduce, you should design your applications to avoid deadlocks and wait conditions.

This chapter discusses how to:

Topics	Page
Understand causes of deadlock conditions	4-2
Understand causes of wait conditions	4-2
Identify deadlocks and wait conditions	4-5
Stop a deadlocked application	4-5

© Copyright IBM Corp. 1994

Understanding Wait Conditions

Understanding Deadlock Situations

Deadlocks may occur when multiple conversations and applications share common resources. If you design an application in which multiple conversations are interleaved, the application is more likely to cause deadlocks.

Some calls that require a response and may cause deadlock are:

Receive_And_Wait.

The following calls may cause a deadlock only when the conversation is half-duplex and queue-level non-blocking is not used.

- Prepare_To_Receive
  This occurs if the conversation was allocated with *sync_level* set to CM_CONFIRM and if the *prepare_to_receive_type* was set to CM_PREP_TO_RECEIVE_SYNC_LEVEL.
- Deallocate
  This occurs if:
  - The conversation was allocated with *sync_level* set to CM_CONFIRM and if the *deallocate_type* was set to CM_DEALLOCATE_SYNC_LEVEL.
  - The *deallocate_type* was set to CM_DEALLOCATE_ABEND.

Understanding CPI-C Wait Conditions

CPI-C programs can wait indefinitely for a partner program or LU to take action. The wait conditions that can occur in both half-duplex and full-duplex conversations are:

Waiting for data

Waiting for an incoming Attach

Waiting for a pacing response

The wait conditions that can occur only in half-duplex conversations are:

Waiting for confirmation

Waiting after an Deallocate or Send_Error call

Waiting for Data

When the *receive_type* is CM_RECEIVE_AND_WAIT, the subsequent Receive call waits indefinitely for data or an indicator to arrive from the partner before returning to the program. If you require continuous operation, try instead issuing a Set_Receive_Type call to set the *receive_type* to CM_RECEIVE_IMMEDIATE and then periodically issuing the Receive call. In that case, the Receive call does not wait for data or an indicator to arrive from the partner. If data or an indicator is available, it is included when the call is returned. If none is available, the returned Receive call indicates so.

Waiting for an Incoming Attach

If your application issues an Accept_Conversation call (CMACCP) and no incoming Attach from a partner is queued, the application waits until either a corresponding incoming Attach arrives or until the Accept_Conversation timeout value configured for that application expires. The timeout can be configured for as long as 16 hours or it can be configured to wait indefinitely by specifying a value of 65 535 for the Accept_Conversation timeout parameter in the TP definition. For more information, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*

Also, if your program issues an Accept_Incoming call (CMACCI) while the *queue_processing_mode* of the Initialization queue is CM_BLOCKING (the default mode), and no incoming Attach from a partner is queued, the application waits until a corresponding incoming Attach arrives. To prevent an indefinite wait, code your program to issue Set_Queue_Processing_Mode (CMSQPM) to CM_NON_BLOCKING for the Initialization queue before it issues CMACCI.

Waiting for Pacing

A Send_Data call waits until the local CPI-C send buffer has enough room for its data.

Pacing prevents one node from flooding its partner with data and using all the available buffers. Pacing in Networking Services is configured using the receive pacing windows parameter in the mode definition. For more information, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

A remote application must issue a Receive call to clear the session send buffers of the partner. If the session send buffers are not cleared, the remote application cannot transmit the pacing response required before the local Send can occur.

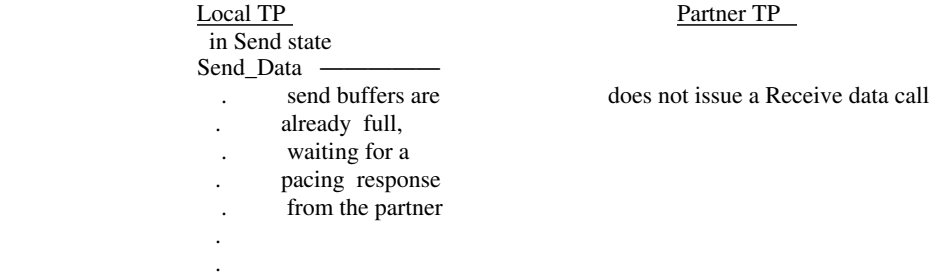

Figure 4-1. Waiting for a Pacing Response

Understanding Wait Conditions

Waiting for Confirmation

This wait condition can occur only in a half-duplex conversation.

If the *queue_processing_mode* of the Send-Receive queue is CM_BLOCKING (the default) and a program issues a Confirm call, CPI-C may wait indefinitely until the remote program issues a valid response.

In addition, any CPI-C call that can include the function of Confirm may wait indefinitely until the remote program issues a Confirmed call. CPI-C calls that can include the Confirm function are:

Deallocate

Prepare_To_Receive

Send_Data

Any of the following actions by the partner application represent a valid response:

Confirmed call

Send_Error call

Deallocate call when *deallocate_type* is set to CM_DEALLOCATE_ABEND

exit(). An exit causes an internal Deallocate call with *deallocate_type* set to CM_DEALLOCATE_ABEND

Any other CPI-C call issued by the partner causes no line flow.

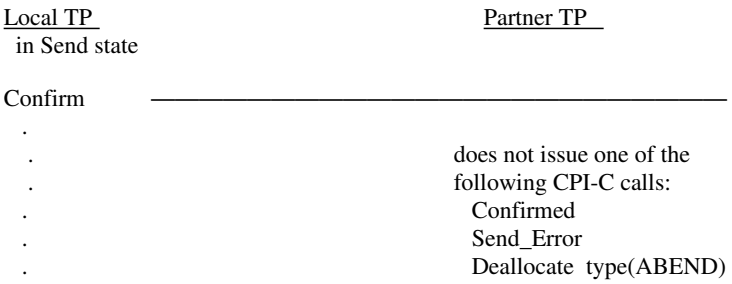

Figure 4-2. Waiting for a Confirmation

Waiting after a Deallocate or Send_Error Call

This wait condition can occur only in a half-duplex conversation.

When a conversation is in Receive state, your application may wait after issuing one of these calls:

Deallocate with type(ABEND)

Send_Error.

In Receive state, Networking Services must wait for the remote computer to send an internal SNA message to the local computer. That flow occurs after the partner conversation on the remote computer issues any conversation call. Thus, if the conversation is active on the partner, but no verb is issued, the local conversation waits indefinitely for the required message.

Identifying a CPI-C Application Deadlock

Networking Services provides a tracing utility to determine when a suspend or wait condition occurs. Tracing indicates both the start and finish of each CPI-C call. This information identifies the last CPI-C call started before a deadlock or wait condition occurred. You can also use tracing to check the parameter values for each CPI-C call as it is executed.

Stopping a Deadlocked CPI-C Program

If a CPI-C program is deadlocked, use the following options to end the program. If you perform one of these functions while a CPI-C verb is being processed, the verb fails with CM_PRODUCT_SPECIFIC_ERROR Then the conversation is deallocated, and the session is deactivated.

Forcibly end the session

If only one conversation is active with the partner LU, do the following:

1. Double-click on the **Advanced Operations** program icon
2. Click on the Operation menu item
3. Select Connections from the pull-down menu
4. Click on the appropriate link in the Active Links list to highlight it
5. Click on the name of the appropriate partner LU in the Partner LUs on Selected Link list to highlight it
6. Click on Sessions in the Additional LU Information group
7. Click on Deactivate
8. Click on OK to force deactivation of all conversations on the session.

Forcibly end all sessions with the partner LU

Do the following:

1. Repeat steps 1 through 5 above.
2. Click on Deactivate in the LU Sessions group
3. Click on OK to force deactivation of all conversations and sessions with that partner LU.

Forcibly end all sessions using the link

Do the following:

1. Repeat steps 1 through 4 under **Forcibly end the session**
2. Click on Disconnect in the Link Control group

Stopping a Deadlocked Program

3. Click on OK to force deactivation of all conversations, sessions, and partner LUs on that link.

Exit Networking Services.

Do the following:

1. Repeat steps 1 and 2 under **Forcibly end the session**
2. Select LU Control
3. Click on Terminate LU
4. Click on OK to force deactivation of all conversations, sessions, partner LUs and links.

Press Ctrl+Alt+Del.

Press Ctrl+Alt+Del to quit an application. Windows displays a message confirming your action. To continue, follow the instructions in the message.

Disconnect the DLC.

If you use an asynchronous or SDLC connection, turn off the modem or disconnect its telephone lead.

Turn Power Off

Turn power off only if none of the other methods work.

Chapter 5. Managing Incoming Attaches

When a program at a remote computer wants to start an APPC conversation with a program at your workstation, it issues a conversation-initiating request to the logical unit 6.2 (LU 6.2) in the remote computer. The LU 6.2 transforms that request into an internal SNA message known as an *Attach* and sends it to your workstation. When the request is received by your workstation, it is called an *incoming Attach*. When a program at your workstation accepts the incoming Attach, an *incoming conversation* starts.

For example, Figure 5-1 shows a program at the remote computer issuing a CPI-C Allocate (CMALLC) call, which is transformed into an Attach. The program at your workstation can issue either an Accept_Conversation (CMACCP) call or an Accept_Incoming (CMACCI) call to accept the incoming conversation. In some cases, Networking Services processes Accept_Conversation and Accept_Incoming calls identically; in those cases the generic term, an *accept*, is used. Where the calls are treated differently, they are named specifically.

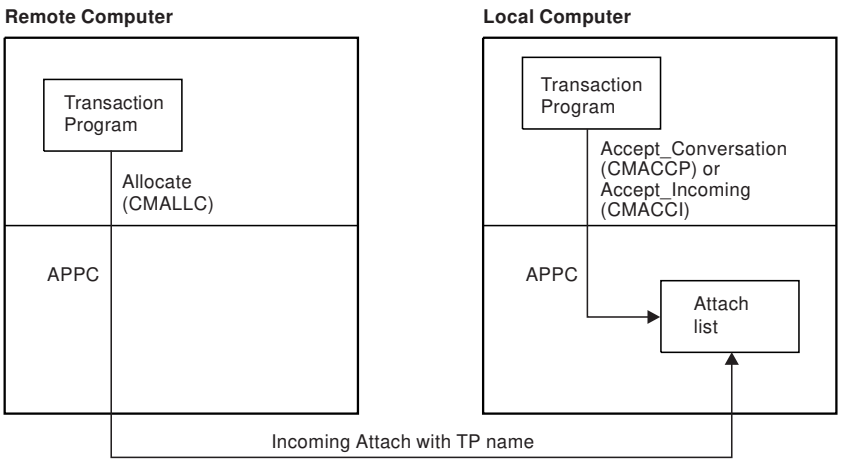

Figure 5-1. The Attach Manager function in Networking Services

This chapter describes the activities that occur when an incoming Attach is received, from the viewpoint of your workstation. Topics discussed include:

Topic	Page
What Is the Attach Manager?	5-2
How Does an Incoming Conversation Start?	5-2
What Information Does the Attach Manager Need?	5-3
Processing Incoming Attaches	5-6
Processing Accepts	5-8

© Copyright IBM Corp. 1994

How Does an Incoming Conversation Start?

Topic	Page
Starting Programs and the Program Launcher	5-12

What Is the Attach Manager?

The Attach Manager is a group of Networking Services functions that:

Manages an *Attach list* of incoming Attaches. The Attach list is a list of all Attaches currently eligible to be accepted by any program that issues a corresponding accept. The Attach list can have multiple entries if multiple sessions are active. Attaches on a given session are processed serially, so that at most one Attach from each session appears in the Attach list. Often only one session is active, and the Attach list has one entry.

Optionally starts programs automatically as a result of an incoming Attach. This function is called the Program Launcher.

How Does an Incoming Conversation Start?

When a remote computer requests a conversation, three events occur at your workstation before the conversation starts:

Networking Services receives an incoming Attach.

The local program starts. You can start the program or you can specify that the Program Launcher automatically start the program when it receives an Attach.

Networking Services receives an accept from the local program.

These events may occur in any order, although the program must be started before it can issue an accept. The following examples illustrate:

An incoming Attach arrives, the Program Launcher starts a program, and the program issues an accept.

The user starts a program, the program issues an accept, and later an incoming Attach arrives.

An incoming Attach arrives, but no program with a name corresponding to the TP name on the Attach is running and the Program Launcher is not active. The Attach Manager keeps the Attach in the Attach list for a timeout period until the user starts a program that issues a corresponding accept. If a corresponding accept is not issued before the timeout expires, the Attach Manager rejects the Attach.

What Information Does the Attach Manager Need?

The Attach Manager needs information to manage incoming Attaches and to automatically start programs on your workstation. The information is provided by *Attach authorization*, which applies to all incoming Attaches, and one or more *TP definitions*. The group of TP definitions is also called the *set of defined TPs*.

For each TP name you expect to receive in an incoming Attach, create a TP definition, consisting of the following required and optional parameters:

Required parameters:

- Transaction program name (TP name)

Optional parameters:

- File specification
- Accept_Conversation timeout
- Incoming Attach timeout
- Transaction program operation (TP operation)
- Parameter string

For more information about the parameters of TP definitions, possible values, and their meanings, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

The following table shows the Attach Manager tasks and the information required to perform each task.

The Attach Manager determines:	By using this information:
Whether to place an incoming Attach in the Attach list	The Attach authorization parameter, and, depending on its value, the TP name in the Attach, and the TP name parameters of the set of defined TPs
How the TP name in an incoming Attach corresponds to a name of a local program and the path on which find the program	The TP name in the incoming Attach and the file specification parameter of the corresponding TP definition
How long to keep the incoming Attach in the Attach list	The incoming Attach timeout parameter of the corresponding TP definition
Whether to automatically start a program when a corresponding incoming Attach is received	The TP operation parameter of the corresponding TP definition

The program parameters parameter of the TP definition is not needed for Attach Manager tasks. However, if the TP operation parameter specifies AM_STARTED, the Attach Manager passes the program parameters string to the program specified by the file specification parameter.

In addition, Networking Services uses the Accept_Conversation timeout parameter of the TP definition to determine how long to wait for a corresponding incoming Attach

What Information Does the Attach Manager Need?

after a local program issues an Accept_Conversation call. For more information about how Accept_Conversation is processed, see “Processing Accepts” on page 5-8.

Provide the Attach authorization and TP definitions in one or more of the following ways:

By using the **Configure** program.

- To provide TP definitions using the **Configure** program:
  1. Double-click on the **Configure** program icon in the IBM APPC Networking Services program group window
  2. Click on Step 3
  3. Click on the option button titled Transaction Programs
  4. Type in the information in the appropriate fields. Also, if you want the program to be operator-started, click on the box titled Started by Program Launcher to remove the X. The transaction program name and file specification fields are required for a TP definition.
  5. Click on OK
- You cannot specify Attach authorization using the **Configure** program.

By using the **Advanced Operations** program.

- To specify Attach authorization:
  1. Double-click on the **Advanced Operations** program icon in the program group window
  2. Click on Operation and select Attach Management from the pull-down menu
  3. Click on the option button for the authorization option you choose.
  4. Click on OK
- To provide TP definitions using the **Advanced Operations** program:
  1. Double-click on the **Advanced Operations** program icon in the program group window
  2. Click on Operation and select Definitions from the pull-down menu
  3. Click on the option button titled Transaction Programs
  4. Type in the information in the appropriate fields. The transaction program name and file specification are required for a TP definition.
  5. Click on OK

  If you attempt to define a TP using **Advanced Operations** program, but do not specify required parameters, you will be notified of an error when you press OK.
- By an application program issuing node operator facility API verbs:

- To specify Attach authorization, an application program issues the verb, NSATTACH.
- To specify a TP definition, an application program issues the verb, NSDEFTP. If an application program issues NSDEFTP without including required parameters, Networking Services returns a return code indicating error.

For more information about node operator API verbs, see Chapter 6, “Using Networking Services Node Operator Application Program Interface” on page 6-1.

If optional parameters are not specified using the methods listed above, Networking Services uses the default values for those parameters. For information about default values, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

Adding to the Attach List

Processing Incoming Attaches

The Attach Manager processes an incoming Attach by:

Deciding whether to add the Attach to the Attach list or to reject it

Calling the Program Launcher so that it can attempt to start the corresponding local program, if so specified by the TP operation parameter.

Removing the Attach from the Attach list and rejecting it after its timeout expires, if it has not been removed from the Attach list by an accept or by other processing.

Adding Incoming Attaches to the Attach List

The Attach Manager decides whether to add an Attach to the Attach list based on the Attach authorization parameter. Upon adding an Attach to the Attach list, the Attach Manager assigns a timeout value to the Attach, as specified:

Attach authorization has one of the following values:

All Attaches All incoming Attaches are added to the Attach list. This is the default option for Attach authorization.

The Attach Manager assigns a timeout values as follows:

If the TP name on the incoming Attach matches a transaction program name parameter in the set of defined TPs, the Attach Manager assigns the value of the incoming Attach timeout parameter of that TP definition to the Attach. If that TP definition has no specification for the incoming Attach timeout parameter, the Attach Manager assigns the default value of 30 seconds.

If the TP name on the incoming Attach does not match the TP name of any entry in the set of defined TPs, the Attach Manager assigns the default value of 30 seconds for the timeout.

Defined Attaches Only

Only incoming Attaches containing a TP name that matches the transaction program name parameter in an entry in the set of defined TPs are added to the Attach list. All others are rejected.

The Attach Manager assigns a timeout value to the Attach as follows: the Attach Manager assigns the value of the incoming Attach timeout parameter of the matching TP definition to the Attach. If that TP definition does not have a specification for the incoming Attach timeout parameter, the Attach Manager assigns the default value of 30 seconds.

No Attaches All incoming Attaches are rejected.

Changing Attach authorization has no effect on any Attaches already in the Attach list.

Calling the Program Launcher

The Program Launcher is an Attach Manager function that attempts to automatically start a program when an Attach is added to the Attach list, unless the Attach corresponds to a TP definition that specifies the TP operation parameter to be OPERATOR_STARTED.

If the Program Launcher is not active when an incoming Attach is added to the Attach list, the Attach remains on this list until its timeout expires or until the Attach is removed by the operator or by an application program. Starting the Program Launcher during that timeout period has no effect on the Attach.

For more information about starting and stopping the Program Launcher, see "Starting and Stopping the Program Launcher" on page 5-12.

Removing an Attach from the Attach List

Once an Attach has been added to the Attach list, the Attach Manager keeps it there until one of the following occurs:

An application program issues an accept that matches the Attach (see Figure 5-2 on page 5-9 and Figure 5-3 on page 5-10)

The Attach is rejected for one of the following reasons:

- The timeout expires. If the associated TP definition contains the maximum value, 65535, for the incoming Attach timeout parameter, the timeout never expires.
- The Program Launcher cannot start the specified program because of an error condition
- You remove the Attach or an application program removes the Attach from the Attach list:
  - An application program removes the Attach by issuing the node operator facility API verb, NSREJATT, specifying the session on which the Attach is queued. For more information about node operator facility API verbs, see Chapter 6, "Using Networking Services Node Operator Application Program Interface" on page 6-1.
  - You can remove the Attach using the Attach Management function of the **Advanced Operations** program. To remove an Attach from the Attach list:
    1. Double-click on the **Advanced Operations** program icon
    2. Click on Operation and select Attach Management from the pull-down menu
    3. Scroll the Attach list until the Attach you want to remove is displayed
    4. Click on Delete

Processing Accepts

Rejecting Incoming Attaches

When Networking Services rejects an Attach, it sends an error description message to the remote computer indicating CM_TP_NOT_AVAILABLE_NO_RETRY.

Processing Accepts

Figure 5-2 on page 5-9 illustrates how Networking Services processes an Accept_Conversation (CMACCP) call, and Figure 5-3 on page 5-10 illustrates how Networking Services processes an Accept_Incoming (CMACCI) call. After Networking Services receives an accept of either type, it continues to process the accept until it sends a return code back to the program that issued the accept.

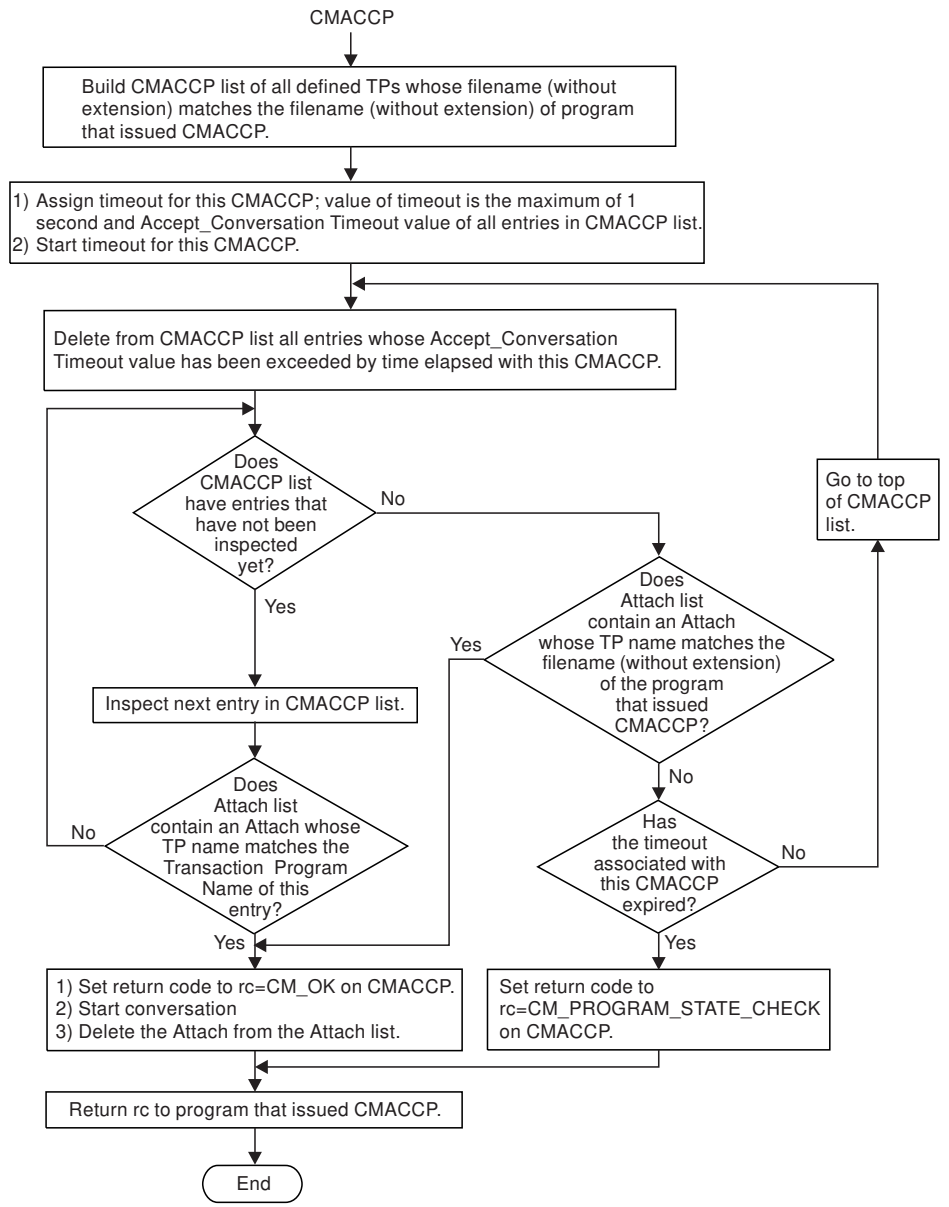

Figure 5-2. Processing an Accept_Conversation

Processing Accepts

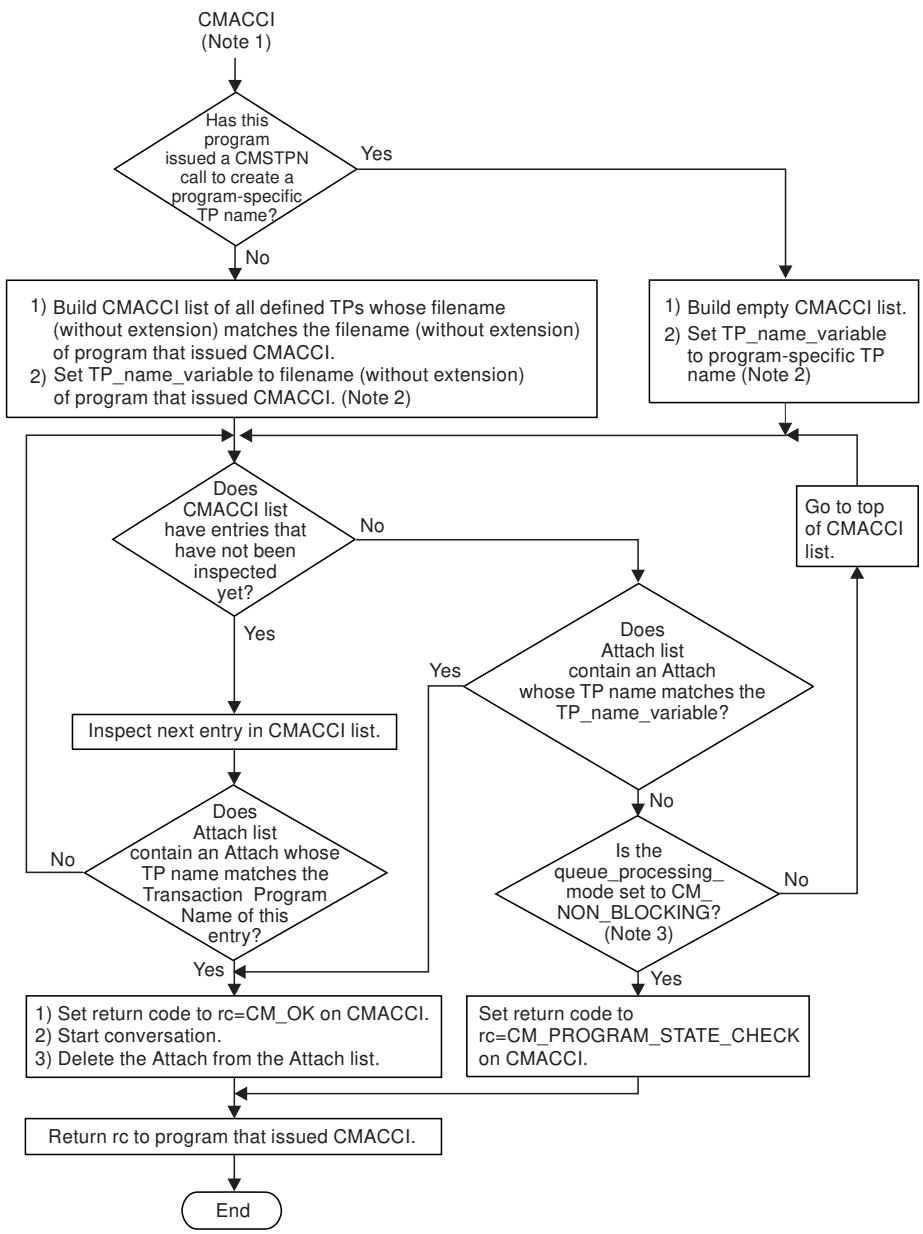

Figure 5-3. Processing Accept_Incoming

Notes:

1. This processing also occurs when a Wait_For_Completion (CMWCMP) call is issued after a CMACCI has been returned with *return_code* set to CM_OPERATION_INCOMPLETE. The processing for Wait_For_Completion calls this logic as a subroutine if Accept_Incoming is one of its outstanding operations.
2. The TP_name_variable is a variable that is used only for this flowchart. It is set either to the program-specific TP name defined by CMSTPN, or to the filename (without extension) of the program that issued the CMACCI.
3. An application program can set the *queue_processing_mode* to CM_NON_BLOCKING by issuing the Set_Queue_Processing_Mode (CMSQPM) call. The default *queue_processing_mode* is CM_BLOCKING, in which case the program's processing is blocked until a corresponding Attach arrives; no timeout of the CMACCI occurs.

Note that the processing of CMACCP and CMACCI is similar except for:

Defining a program-specific TP name:

Before a program issues CMACCI, it can issue the Set_TP_Name (CMSTPN) call to associate a program-specific TP name with itself. When a program-specific TP name has been so defined, Networking Services matches an Attach to a CMACCI only if the TP name on the Attach matches the program-specific TP name.

Note that issuing a CMSTPN call defines only a program-specific TP name. This TP is not defined for any other program.

If the program has not issued CMSTPN, no program-specific TP name definition exists. In this case, the Attach Manager uses the set of defined TPs exactly as it does for CMACCP.

Deciding how long to wait for a matching incoming Attach:

- When Networking Services processes a CMACCP, it uses the Accept_Conversation timeout parameters in the set of TP definitions to determine how long to wait for a matching incoming Attach.
- Before a program issues CMACCI, it can issue the call, Set_Queue_Processing_Mode (CMSQPM) to set the *queue_processing_mode* of the Initialization queue. If a program issues a CMACCI when this mode is set to CM_BLOCKING (the default mode), Networking Services waits until a corresponding Attach arrives; no timeout of the CMACCI occurs. If a program issues a CMACCI when this mode is set to CM_NON_BLOCKING, and no corresponding Attach is currently in the Attach list, the CMACCI returns with a return code set to CM_OPERATION_INCOMPLETE. This allows the program to continue with other processing. Later, the program can issue a Wait_For_Completion (CMWCMP) call with CMACCI as one of its outstanding operations.

Starting Programs and the Program Launcher

Programs are either started by the user or started automatically by the Program Launcher, if it is active.

Starting Programs

If the Program Launcher has been started, it automatically starts a program when the Attach Manager receives an Attach whose TP name corresponds to a TP definition that has the transaction program operation parameter specified as AM_STARTED. After the Program Launcher has started a program, the Program Launcher has no further interaction with the program.

If you do not want more than one instance of your program to run at a time, do one of the following:

Have only one TP definition for a given TP name, and specify OPERATOR_STARTED in the TP definition.

For AM_STARTED programs, write the program so that it immediately ends a second instance of itself. This can be done by inspecting the *hPrevInstance* parameter of *WinMain*. If this parameter has a nonzero value, a previous instance exists, and the program should return from *WinMain*.

Starting and Stopping the Program Launcher

The Program Launcher operation parameter in the network configuration file, NSD.INI, specifies whether the Program Launcher is to be started when Networking Services is started. If you do not want the Program Launcher to start when Networking Services starts, edit NSD.INI to specify FALSE for this parameter.

You can also use the **Advanced Operations** program to start and stop the Program Launcher.

To start or stop the Program Launcher:

1. Double-click on the **Advanced Operations** program icon
2. Click on Operation and select Attach Management from the pull-down menu
3. In the Attach Management window, if an X appears in the box titled Run Program Launcher, the Program Launcher is active. If the X does not appear, the Program Launcher is not active. If the box does not show the activity status that you want, click on the box to change the status.

Part 3. Using the Node Operator Facility Application Program Interface

Chapter 6. Using Networking Services Node Operator Application Program Interface 6-1
Verb Data Structures 6-1
Return Values and Fields 6-2
Define Symbolic Constants 6-4
NSACT 6-6
NSATTACH 6-10
NSCONLAN 6-11
NSCONMPT 6-13
NSCONREM 6-14
NSCONTWX 6-16
NSDEACT 6-17
NSDEASES 6-18
NSDEFMO 6-19
NSDELMO 6-20
NSDEFSI 6-21
NSDELSI 6-23
NSDEFTP 6-24
NSDELTP 6-26
NSDISCON 6-27
NSINFO 6-28
NSREJATT 6-32
NSSTART 6-33
NSSTOP 6-34

© Copyright IBM Corp. 1994

Chapter 6. Using Networking Services Node Operator Application Program Interface

Networking Services provides a node operator facility application program interface (API) that allows an application program to:

- Start and stop Networking Services
- Connect and disconnect links
- Activate and deactivate partner logical units (LUs)
- Start and stop sessions
- Add or delete mode, side information, and TP definitions
- Change the authorization of which incoming Attaches are to be added to the Attach list
- Remove an Attach from the Attach list
- Display current status

To use the node operator facility API, an application program issues one of the API verbs, which are defined in this chapter.

Verb Data Structures

The data structures described in this chapter are excerpts from the header file, NOFNSD.H. They are used as examples only in this chapter. To include any data structure in your application program, use NOFNSD.H.

The data structure for each API verb contains input and output parameters:

- Input parameters are identified with I as the first character in the comments for that parameter; the possible values and a brief description of each value are also included the comments. Some input parameters are equivalent to keywords that are used in the NSD.INI file; comments for those parameters indicate the equivalent keyword. For more information about keywords, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*. Input parameters are returned unaltered when the verb is returned.
- Output parameters are identified with O as the first character in the comments for that parameter. Two output parameters, ReturnCode and WhichParmError, are common to all verbs; some verbs have additional output parameters.

All input fields should be either the correct value or null (\0). Strings must be null terminated unless otherwise indicated. All string values are taken and returned in ASCII.

© Copyright IBM Corp. 1994

Notes for Compiling Your Program:

The fields in the data structure are packed.

Enumerations are two bytes in length.

When you compile your program, you must use the appropriate compiler switches to pack the fields and specify enumeration length. For example:

If you use the Microsoft C++ compiler, use **-Zp1**

If you use the Borland C++ compiler, use **-a-** and **-b** (both are default switch settings)

Return Values and Fields

After Networking Services processes a verb, it returns a functional value and also returns output parameters in the data structure.

Functional Value

The functional value is returned separately from the output parameters of a data structure. The functional value indicates whether Networking Services was able to handle the issued verb.

The following functional values can be returned:

0 The verb processing and error checking proceeded normally. This value indicates that output parameters contained in the data structure contain meaningful data. This value does not imply that the action requested by the verb was performed. For example, the output parameters, ReturnCode and WhichParmError, may indicate that an error has prevented the requested action from being performed.

8 An invalid control block passed. For example, the control block pointer is null, or the pointer is a code rather than a data pointer. This error prevents further processing of the verb. The settings of all output parameters are meaningless.

Other Returned Error Indicators

When a verb has been returned with a functional value of zero, the following output parameters indicate whether errors have been detected:

ReturnCode The first field in each verb's data structure indicates whether Networking Services was able to perform the action requested by the verb. A zero return code indicates that the desired action was performed. A nonzero return code usually indicates that a parameter is in error, or that Networking Services is not loaded or available. For some verbs, the most common return codes are shown in the comments. All possible return codes for each verb are listed in the file, NOFCODES.DOC in the INCLUDE subdirectory of the directory where Networking Services is installed.

General Return Codes

The ReturnCode output parameter can have one of the following values:

X'5xxx' or X'8xxx' To display an explanation of the probable cause and suggested recovery for these return codes, do the following:

1. Double-click on the **Advanced Operations** icon in the APPC Networking Services program group
2. Click on the Help action bar item
3. Click on Contents from the pull-down menu
4. Click on Error Messages
5. Scroll down to the number for which you need information and click on it.

X'50' This return code occurs only on the NSACT verb. It indicates that an allocate error has occurred; check the AllocateRC output parameter for more information. For more information about possible values for AllocateRC, see “Allocate Return Codes” on page 6-8.

WhichParmError The second field in each verb's data structure indicates whether one or more input parameters were found to be in error. This field is represented as bit flags; each bit corresponds to certain input parameters as defined in the *#define* statements that precede the data structure. In most cases, input parameters and bit flags have a one-to-one correspondence. However, some input parameters share a bit flag, and some input parameters have no corresponding bit flag.

A zero value indicates that all input parameters were successfully parsed and passed applicable checks.

A nonzero value indicates which input parameters were found to be in error. For example, if a WhichParmError value of X'15' is returned on an NSDEFMO verb, then the Version, MaxRU, and MaxNegSess parameters were in error. Networking Services attempts to check all input parameters; however, sometimes it is not possible to perform all checks.

Note: It is not possible to get both a zero ReturnCode and nonzero WhichParmError.

Other Returned Parameters

The verbs, NSACT and NSINFO, also have additional output parameters for information.

Define Symbolic Constants

The following statements are used to define constants that are not related to parameter errors or type definitions.

```
/ First, a definition that your program may need                    /
#ifndef BYTE
   #define BYTE unsigned char
#endif

/ the following value is returned by the function implementing a NOF    /
/ verb when a NULL or invalid pointer is passed in (an example of an    /
/ invalid pointer is one that does not have both read and write access) /
#define INVALID_CONTROL_BLOCK_POINTER  x 8

#define DISCON_ALL_LINKS " ALL"    / see the Disconnect_Link structure   /
#define DEACT_ALL_LUS      " ALL"    / see the Deactivate_LU structure     /

#define MAX_LINKS 6          / maximum number of LAN links                 /

/ string lengths /
#define MODE_LEN 9           / 8 + null char                               /
#define SYMD_LEN 9           / 8 + null char                               /
#define LUN_LEN   18         / 8 + '.' + 8 + null char                    /
#define FILE_LEN 65         / (DOS) 64 + null char                       /
#define TPN_LEN   132        / 'X' + single quote + 2   64 (for           /
                             / hex chars) + single quote + null          /
#define TPN_LEN_OUT 65     / 64 chars + null                             /
#define  PARM_LEN    129   /  128  +  null  char                           /

#define SAP_LEN 3            / 2 chars + null                             /
#define IDNUM_LEN 4          / 4 bytes (no null)                          /
#define LINK_LEN 9           / 8 + null char                              /
#define USERID_LEN 11      / 1  + null char                               /
#define PASSWORD_LEN 11   / 1  + null char                                /

#define LAN_ADDR_LEN 13   / 12 chars + null                               /
#define SWAD_LEN         62
#define FQPCID_LEN     17   / 16 chars + null                             /

/ typedef and values for Conversation Security for NSDEFSI call /
/      and typedef Define_Side_Info_Entry                          /
```

```
typedef unsigned char ConversationSecurity;
#define SECURITY_NONE
#define SECURITY_SAME        1
#define SECURITY_PROGRAM 2
```

NSACT

Use this verb to create a partner LU definition and to optionally activate one or more sessions with a specified LU. You can also use this verb to:

- Specify a minimum number of sessions to remain active
- Specify a link for routing communications to an LU
- Specify a mode for a session
- Specify a user ID and password to be used for security on conversations that use the sessions

You can also use this verb to create a partner LU definition without activating any user sessions. If you issue this verb with only the link and LU parameters specified, the blank mode is assumed. For more information about partner LU definition, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

Output Parameters

In addition to the return codes shown on page 6-3, the NSACT verb can return the following output parameters:

AllocateRC	Allocate return code. Check this field if ReturnCode=50. See list below for possible Allocate return codes and their meanings.
SenseCode1	SNA sense code. Check this field if AllocateRC is not 0. SenseCode1 and SenseCode2 form SNA sense data.
SenseCode2	SNA sense-code-specific information.
ActivatedSessions	Number of sessions activated as a result of this call to the API Check this field if AllocateRC=0.
TotalUserSessions	Total number of sessions currently active. This number is equal to the number of sessions active before the call to the API plus the value of ActivatedSessions. Check this field if AllocateRC=0.
AutoActivateLimit	Current value for how many sessions should be automatically activated. This value is equal to the input parameter, MinSessions. Check this field if AllocateRC=0.

Allocate Return Codes

The following table shows the possible Allocate return codes, the Networking Services error codes to which they are equivalent, and their meanings. For additional information about possible cause and recovery, follow the directions on page 6-3.

AllocateRC	Equivalent Error Code	Meaning
0		Command completed successfully
1	8841	Session not activated because no links are active
2	8842	No storage is available in buffer pool
3	8843	ModeName not defined
4	8844	CNOS failed
5	8845	Session limit reached
6	8846	Parallel sessions not allowed
7	8847	Single session not allowed
8	8848	Bind reject - Check SenseCode1 and SenseCode2 **(Note)**
9	8849	Attach reject - Check SenseCode1 and SenseCode2
10	Reserved	
11	8851	CNOS failed - mode name not in remote
12	8852	CNOS failed - remote refused
13	8853	CNOS failed due to race
14	8854	DLC send failed
15	8855	Remote computer sent UNBIND - Check SenseCode1 and SenseCode2
16	Reserved	
17	8867	Session outage due to terminate - deactivation
18	8866	Session outage due to link failure
19	8868	Partner LU name same as local LU name
Other		Check SenseCode1 and SenseCode2

Note: If SenseCode1=X'0000', the partner LU sent a negative response to BIND, but no further information is available. A commonly used SenseCode1 is X'8004', indicating that the partner LU could not be found. SenseCode2 can provide further information to indicate the source of the problem.

Function Prototype

```
extern int pascal far _loadds NSACT(Activate_LU far );
```

Data Structure

```
#define ActivateLUVersionParmError  x   1
#define ActivateLULuNameParmError  x   2
#define ActivateLUModeNameParmError  x   4
#define ActivateLULinkNameParmError  x   8
#define ActivateLUMinSessionsParmError  x  1
#define ActivateLUUserIDParmError  x  2
#define ActivateLUPasswordParmError  x  4
#define ActivateLUDefineOnlyParmError  x  8

typedef struct activate_lu
{
        unsigned int      ReturnCode;             /  O-Zero if ok, else error                  /
                                                  /       5 =Allocate error, see AllocateRC /
                                                  /       other - see general definition      /
        unsigned int      WhichParmError;         /  O-this indicates which input             /
                                                  /       parm below is in error                /
                                                  /       uses the flags defined above        /
        unsigned int      AllocateRC;             /  O-Allocation return code;zero if ok /
                                                  /       else error, see list above          /
        unsigned int      SenseCode1;             /  O-if Allocate RC <>                      /
        unsigned int      SenseCode2;             /  O-if Allocate RC <>                      /
        unsigned int      ActivatedSessions; /  O-number of sessions activated         /
        unsigned int      TotalUserSessions; /  O-total number of user sessions       /
        unsigned int      AutoActivateLimit; /  O-auto activate limit                  /
        unsigned char     Version;                /  I-  indicates MinSessions is last    /
                                                  /          field,                                 /
                                                  /       1 indicates UserID, Password,       /
                                                  /          DefineOnly,  reserved1,             /
                                                  /          reserved2, and reserved3 are     /
                                                  /          included                                /
        char                LuName[LUN_LEN];      /  I-Name of LU to activate:              /
                                                  /       netname.luname or luname (netname /
                                                  /       defaults to netname of local LU)  /
        char                ModeName[MODE_LEN];/  I-mode name to use for LU             /
        char                LinkName[LINK_LEN];/  I-link name for the LU,               /
                                                  /       null=use default link               /
        int                 MinSessions;          /  I-how many sessions: -1=single;       /
                                                  /       1-99=number of parallel             /
                                                  /       sessions,  =add LU+mode only        /
        /  the following three fields are present only if Version = 1                 /
        char                UserID[USERID_LEN];/  I-User ID, if this field is blank,  /
                                                  /       then UserID and password are        /
                                                  /       ignored                                 /
       char                 Password[PASSWORD_LEN];/   I-password                          /
        char                DefineOnly;           /  I- =activate sessions (send CNOS);  /
                                                  /       non-zero: define the LU only        /
                                                  /  Note: if this field is set, then       /
                                                  /  MinSessions must be   and ModeName  /
                                                  /  must be "", else WhichParmError        /
                                                  /  points to this (DefineOnly) field     /
        void far          reserved1;              /  reserved                                /
       int                 reserved2;             /   reserved                               /
       unsigned  int      reserved3;              /   reserved                               /
} Activate_LU;
```

NSATTACH

Use this verb to specify the Attach authorization. For more information about how Networking Services uses the value of Attach authorization, see “Processing Incoming Attaches” on page 5-6.

Function Prototype

```
extern int pascal far _loadds NSATTACH(Attach_Authorization far );
```

Data Structure

```
#define AttachAuthorizationVersionParmError x 1
#define AttachAuthorizationAuthorizationParmError x 2

typedef struct attach_authorization
{
        int                 ReturnCode;          / return code                          /
        unsigned int     WhichParmError;      / O-this indicates which input      /
                                                  /     parm below is in error          /
                                                  /     uses the flags defined above   /
        unsigned char    Version;             / I-must be                            /
        int                 Authorization;      / I-  Authorize all attaches        /
                                                  /     1 Authorize defined only        /
                                                  /     2 Authorize none                /
} Attach_Authorization;
```

NSCONLAN

Use this verb to activate a LAN link. For more details about the input parameters, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

Function Prototype

```
extern int pascal far _loadds NSCONLAN(Open_LAN_Station far );
```

Data Structure

```
#define OpenLanStationVersionParmError x 1
#define OpenLanStationLinkNameParmError x 2
#define OpenLanStationRemAddrParmError x 4
#define OpenLanStationRemSAPParmError x 8
#define OpenLanStationConnRetryParmError x 1
#define OpenLanStationConnTimerParmError x 2
#define OpenLanStationWaitForRspTimerParmError x 4
#define OpenLanStationRcvAckSendTimerParmError x 8
#define OpenLanStationInactLinkTimerParmError x 1
#define OpenLanStationMaxIFrameOutParmError x 2
#define OpenLanStationMaxIFrameInParmError x 4
#define OpenLanStationFrameRetryParmError x 8

typedef struct open_lan_station
{
    int             ReturnCode;          / O-Zero if ok, else error                /
                                         /     5148=Network is beaconing;          /
                                         /     52 3=Link name is duplicated;       /
                                         /     other - see general definition      /
    unsigned int    WhichParmError;      / O-this indicates which input            /
                                         /     parm below is in error              /
                                         /     uses the flags defined above        /
    unsigned char   Version;             / I-must be   or 1                        /
                                         /       - ConnTimer is the last field     /
                                         /     1 - Includes WaitForRspTimer,       /
                                         /           RcvAckSendTimer,InactLink-    /
                                         /           Timer, MaxIFrameOut, MaxI-    /
                                         /           FrameIn, FrameRetry, and      /
                                         /           IDNum                         /
                                         / Following fields are equivalent       /
                                         / to parameters on TRLD.   For more     /
                                         / information, refer to Application     /
                                         / Programmer's Reference                /
    char            LinkName[LINK_LEN];/ I-Link name                             /
    char            RemAddr[LAN_ADDR_LEN];
                                         / I-LAN Address of the remote; must     /
                                         /     must be 12 hex chars              /
    char            RemSAP[SAP_LEN];     / I-Service Access Point in remote;     /
                                         /     use "" or " " or "  " for         /
                                         /     default; must be a multiple of 4  /
    int             ConnRetry;           / I-# frame retries for connect;        /
                                         /      -255 are valid values; use -1    /
                                         /     for  default                      /
    int             ConnTimer;           / I-this value times ConnRetry yields /
                                         /     number of half-seconds before     /
```

```
                                        /      timing out.    1-255 are valid          /
                                        /      values; use -1 or   for default     /
        /  the following fields are available if Version = 1                          /
        unsigned char    WaitForRspTimer;      /  I-wait time after send frame,            /
                                        /      between   and 1 , inclusive;          /
                                        /        to use default                       /
                                        /           (T1 / response timer on TRLD)  /
        unsigned char    RcvAckSendTimer;      /  I-wait time (local) before sending  /
                                        /      ACK to received frame, between       /
                                        /        and 99, inclusive;                  /
                                        /        to use default                      /
                                        /  (T2 /acknowledgement timer on TRLD) /
        unsigned char    InactLinkTimer;       /  I-link inactivity timer, between       /
                                        /        and 1 , inclusive;                  /
                                        /        to use default                      /
                                        /        (Ti / inactivity timer on TRLD)  /
        int              MaxIFrameOut;          /  I-maximum number of sent I-frames     /
                                        /      outstanding, between   and 127,     /
                                        /      inclusive; or -1 for default          /
                                        /      (MaxXmitsNoRcv / MAXOUT on TRLD)  /
        int              MaxIFrameIn;           /  I-maximum number of received I-        /
                                        /      frames before sending ACK,              /
                                        /      between   and 127, inclusive;        /
                                        /      or -1 for default                       /
                                        /      (MaxRcvsNoXmit /    MAXIN on TRLD)  /
        unsigned char    FrameRetry;            /  I-number of frames re-transmitted     /
                                        /      without ACK receipt before link     /
                                        /      is dropped;   means 8 attempts       /
        BYTE             IDNum[IDNUM_LEN];    /  I-8 hex chars representing the          /
                                        /      block id/num--this field is an       /
                                        /      array of bytes,  not  a string;     /
                                        /      set all bytes to  xFF to use          /
                                        /      value specified by the NODEID        /
                                        /      keyword in the nsd.ini file           /
} Open_LAN_Station;
```

NSCONMPT

Use this verb to activate a link that uses Multi-Protocol Transport Networking. For more details about the input parameters, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers* and refer to the keyword or parameter shown in the comments field.

Function Prototype

```
extern int pascal far _loadds NSCONMPT(Open_MPTN far );
```

Data Structure

```
#define OpenMPTNVersionParmError x 1
#define OpenMPTNLinkNameParmError x 2

typedef struct open_mptn {
    int             ReturnCode;          / O-Zero if ok; else error                /
                                         /    52 3=Link name duplicated;           /
                                         /    other - see general definition       /
    unsigned int    WhichParmError;      / O-this indicates which input            /
                                         /    parm below is in error                /
                                         /    uses the flags defined above          /
    unsigned char   Version;             / I-must be                                /
    char            LinkName[LINK_LEN];/ I-Link name                    (MPLD) /
} Open_MPTN;
```

NSCONREM

Use this verb to activate an SDLC or an SDLC over asynchronous (ASYNC) link. For more details about the input parameters, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers* and refer to the keyword or parameter shown in the comments field.

Notes on WhichParmError

This verb has more parameters than can be mapped one-to-one with the bits in WhichParmError. The following fields are either mapped together to one bit in WhichParmError or need no flag because an input error is not possible:

The parameter error flag for TimeOut and ConnectTimer is shared.

The fields IDNum and SWAD do not have error flags

Function Prototype

```
extern int pascal far _loadds NSCONREM(Open_REMOTE far  );
```

Data Structure

```
#define OpenRemoteVersionParmError  x   1
#define OpenRemoteLinkNameParmError  x   2
#define OpenRemoteStationAddressParmError  x 1
#define OpenRemoteStationRoleParmError  x 2
#define OpenRemoteTimeOutParmError  x 4
#define OpenRemoteConnectTimerParmError  x 4

typedef struct open_remote
{
        int                 ReturnCode;             /  O-Zero if ok; else error                  /
                                                    /      52 3=Link name duplicated;            /
                                                    /      other - see general definition      /
        unsigned int     WhichParmError;       /  O-this indicates which input             /
                                                    /      parm below is in error                /
                                                    /      uses the flags defined above          /
        unsigned char    Version;                /  I-must be                                  /
                                                    /  Following fields are equivalent to  /
                                                    /  parameters in network configuration /
                                                    /  file.    For more information, refer  /
                                                    /  to keyword or parameter (shown in     /
                                                    /  parentheses) in Application              /
                                                    /  Programmer's Reference                    /
        char                LinkName[LINK_LEN];/  I-Link name                          (SDLD)  /
        unsigned char    StationAddress;       /  I- x 1 thru  xFE are valid    (SDLD)  /
        unsigned char    TimeOut;                /  I-3-4 ,   to use default                  /
                                                    /      (frame recovery timeout on SDLD)  /
        unsigned  char   NumberOfTimeouts;   /   I-1-255,     to  use  default                /
                                                    /        (frame recovery number on SDLD)  /
        unsigned char    ConnectTimer;          /  I-5-255,   to use default                 /
                                                    /             (connection timeout on SDLD)  /
        unsigned char    StationRole;           /  I- -sec 1-pri 2-negotiable    (SDLD)  /
        unsigned char    ActivateRole;          /  I- -caller 1-called; other values     /
                                                    /          will use caller              (SDLD)  /
        BYTE                IDNum[IDNUM_LEN];    /  I-8 hex chars representing the          /
                                                    /      block id/num--this field is an       /
                                                    /      array of bytes,  not  a string       /
                                                    /      set all bytes to  xFF to use          /
                                                    /      value specified by the NODEID        /
                                                    /      keyword in the nsd.ini file           /
        char                SWAD[SWAD_LEN];       /  I-Number to be auto-dialed                /
                                                    /                    (telephone  number)          /
} Open_REMOTE;
```

NSCONTWX

Use this verb to activate an twinaxial link. For more details about the input parameters, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers* and refer to the keyword or parameter shown in the comments field.

Function Prototype

```
extern int pascal far _loadds NSCONTWX(Open_TWINAX far  );
```

Data Structure

```
#define OpenTwinaxVersionParmError  x   1
#define OpenTwinaxLinkNameParmError  x   2
#define OpenTwinaxWorkStationAddressParmError  x   4

typedef struct open_twinax
{
        int                 ReturnCode;             /  O-Zero if ok; else error                  /
                                                      /      52 3=Link name duplicated;            /
                                                      /      other - see general definition      /
        unsigned int     WhichParmError;       /  O-this indicates which input              /
                                                      /      parm below is in error                /
                                                      /      uses the flags defined above          /
        unsigned char    Version;                /  I-must be                                    /
                                                      /  Following fields are equivalent to  /
                                                      /  parameters in network configuration /
                                                      /  file.    For more information, refer  /
                                                      /  to keyword (shown in parentheses)     /
                                                      /  in User's Guide                            /
        char                LinkName[LINK_LEN];/  I-Link name                          (EMLD)  /
        BYTE                  WorkStationAddress;/  I-valid values:  -6              (EMLD)  /
} Open_TWINAX;
```

NSDEACT

Use this verb to deactivate a partner LU, or to deactivate all partner LUs. When a partner LU is deactivated, all sessions with it are deactivated, and the partner LU definition is discarded.

The force parameter, if set to nonzero, will cause the partner LU to be deactivated even if conversations with the partner LU are active.

Function Prototype

```
extern int pascal far _loadds NSDEACT(Deactivate_LU far );
```

Data Structure

```
#define DeactivateLUVersionParmError  x   1
#define DeactivateLULuNameParmError  x   2

typedef struct deactivate_lu
{
      unsigned int     ReturnCode;            / O-Zero if ok, else error              /
                                              /      5148=Network is beaconing        /
                                              /      other - see general definition  /
      unsigned int     WhichParmError;        / O-this indicates which input         /
                                              /      parm below is in error           /
                                              /      uses the flags defined above     /
      unsigned char    Version;               / I-must be                            /
      char             LuName[LUN_LEN];       / I-Name of LU to deactivate           /
                                              /      or use DEACT_ALL_LUS to remove  /
                                              /      all partner LU definitions,     /
                                              /      respecting the force parameter  /
      int              Force;                 / I-nonzero=force,  =do not force     /
} Deactivate_LU;
```

NSDEASES

Use this verb to deactivate a session specified by number. To determine the numbers assigned to active sessions, use the verb, NSINFO.

The force parameter, if set to nonzero, will cause the session to be deactivated even if conversations using the session are active.

Function Prototype

```
extern int pascal far _loadds NSDEASES(Deactivate_Session far );
```

Data Structure

```
#define DeactivateSessionVersionParmError  x   1
#define DeactivateSessionSessionNumberParmError  x   2

typedef struct deactivate_session
{
        unsigned int      ReturnCode;           / O-Zero if ok, else error                /
                                                /      5148=Network is beaconing          /
                                                /      other - see general definition  /
        unsigned int      WhichParmError;       / O-this indicates which input          /
                                                /      parm below is in error             /
                                                /      uses the flags defined above       /
        unsigned char     Version;              / I-must be                              /
        unsigned int      SessionNumber;        / I-number of the session to             /
                                                /      deactivate.    This number can     /
                                                /      be obtained through the NSINFO  /
                                                /      verb                                /
        int               Force;                / I-nonzero=force,  =do not force     /
} Deactivate_Session;
```

NSDEFMO

Use this verb to define a mode after Networking Services has been started.

A mode defined by NSDEFMO is not added to NSD.INI. When you stop Networking Services, the mode defined by NSDEFMO is no longer defined.

If this verb is issued with a mode name that has already been defined, the verb fails, and a nonzero ReturnCode is returned.

The input parameters of NSDEFMO are equivalent to the parameters of an entry in NSD.INI. For more information about the input parameters, refer to *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

Function Prototype

```
extern int pascal far _loadds NSDEFMO(Define_Mode far );
```

Data Structure

```
#define DefineModeVersionParmError  x   1
#define DefineModeModeNameParmError  x   2
#define DefineModeMaxRUParmError  x   4
#define DefineModeRcvPacingWinParmError  x   8
#define DefineModeMaxNegSessParmError  x  1
#define DefineModeMinNegConWinParmError  x  2

typedef struct define_mode {
   unsigned int     ReturnCode;          / O-Zero if ok, else error                  /
   unsigned int     WhichParmError;      / O-this indicates which input              /
                                         /     parm below is in error                /
                                         /     uses the flags defined above          /
   unsigned char    Version;             / I-must be                                 /
   char               ModeName[MODE_LEN];/ I-mode name, CPSVCMG not allowed      /
   unsigned int     MaxRU;               / I-maximum RU size,   indicates           /
                                         /     LU will compute default based on    /
                                         /     max link frame size                 /
                                         /     values between 256 and 16384 ok     /
   unsigned int     RcvPacingWin;        / I-receive pacing window                  /
                                         /     values between 1 and 63             /
   unsigned int     MaxNegSess;          / I-maximum negotiable sessions            /
                                         /     up to 32767                         /
                int     MinNegConWin;    / I-minimum negotiable conwinners          /
                                         /     this field is optional, set to      /
                                         /     -1 to use the default               /
} Define_Mode;
```

NSDELMO

Use this verb to delete a mode definition after Networking Services has been started. A mode deleted by NSDELMO is not deleted from NSD.INI. The SNASVCMG mode may not be deleted.

If this verb is issued with a mode name that does not exist in the mode definitions table, the verb fails, and a nonzero ReturnCode is returned.

Function Prototype

```
extern int pascal far _loadds NSDELMO(Delete_Mode far );
```

Data Structure

```
#define DeleteModeVersionParmError  x   1
#define DeleteModeModeNameParmError  x   2

typedef struct delete_mode {
   unsigned int     ReturnCode;          /  O-Zero if ok, else error                  /
   unsigned int     WhichParmError;      /  O-this indicates which input              /
                                         /     parm below is in error                 /
                                         /     uses the flags defined above          /
   unsigned char    Version;             /  I-must be                                 /
   char               ModeName[MODE_LEN];/  I-mode name, SNASVCMG not allowed     /
                                         /     if SNASVCMG, WhichParmError set     /
} Delete_Mode;
```

NSDEFSI

Use this verb to define a side information definition after Networking Services has been started.

Side information defined by NSDEFSI is not added to NSD.INI. When you stop Networking Services, the side information defined by NSDEFSI is no longer defined.

If this verb is issued with a SymDestName that has already been defined, the verb fails, and a nonzero ReturnCode is returned. Also, the blank SymDestName cannot be defined.

For more information about the input parameters, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

Function Prototype

```
extern int pascal far _loadds NSDEFSI(Define_Side_Info_Entry far );
```

NSDEFSI

Data Structure

```
#define DefineSideInfoVersionParmError  x   1
#define DefineSideInfoSymDestNameParmError  x   2
#define DefineSideInfoPLUNameParmError  x   4
#define DefineSideInfoModeNameParmError  x   8
#define DefineSideInfoTPNameParmError  x  1
#define DefineSideInfoConvSecurityParmError  x  2

typedef struct define_side_info_entry {
   unsigned int     ReturnCode;          /  O-Zero if ok, else error                  /
   unsigned int     WhichParmError;      /  O-this indicates which input              /
                                         /      parm below is in error                /
                                         /      uses the flags defined above         /
   unsigned char    Version;             /  I-Zero if TPName is last field;       /
                                         /      One if SecurityType, UserID,         /
                                         /      Password, and Reserved1               /
  char              SymDestName[SYMD_LEN];
                                         /  I-side information entry name          /
  char              PartnerLUName[LUN_LEN];
                                         /  I-partner LU name                          /
   char              ModeName[MODE_LEN];/  I-mode name                                    /
   char              TPName[TPN_LEN];     /  I-transaction program name, if         /
                                         /      TPName = " ", then TPName will      /
                                         /      be set to "" (empty string)          /
  /  the next four fields are present iff Version = 1                                /
  ConversationSecurity
                     ConvSecurity;       /  I-Conversation Security type, see     /
                                         /      constants  section                    /
   char              UserID[USERID_LEN];/  I-UserID for security                     /
  char               Password[PASSWORD_LEN];
                                         /  I-Password for security                  /
  char               Reserved1[9];       /   reserved                                 /
} Define_Side_Info_Entry;
```

NSDELSI

Use this verb to delete a side information definition after Networking Services has been started. Side information definitions deleted by NSDELSI is not deleted from NSD.INI.

If this verb is issued with a SymDestName that has not been defined, the verb fails, and a nonzero ReturnCode is returned.

Function Prototype

```
extern int pascal far _loadds NSDELSI(Delete_Side_Info_Entry far  );
```

Data Structure

```
#define DeleteSideInfoVersionParmError  x   1
#define DeleteSideInfoSymDestNameParmError  x   2

typedef struct delete_side_info_entry {
   unsigned int     ReturnCode;          / O-Zero if ok, else error                /
   unsigned int     WhichParmError;      / O-this indicates which input            /
                                         /     parm below is in error                /
                                         /     uses the flags defined above        /
   unsigned char    Version;             / I-must be                               /
  char               SymDestName[SYMD_LEN];
                                         / I-side information entry name          /
} Delete_Side_Info_Entry;
```

NSDEFTP

Use this verb to create a TP definition after Networking Services has been started.

TP definitions created by NSDEFTP are not added to NSD.INI. When you stop Networking Services, the TP definitions created by NSDEFSI are no longer defined.

If this verb is issued with a TPName that has already been defined, the verb fails, and a nonzero ReturnCode is returned.

For more information about the input parameters, refer to the *IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers*.

Function Prototype

```
extern int pascal far _loadds NSDEFTP(Define_TP far  );
```

Data Structure

```
#define DefineTPVersionParmError  x   1
#define DefineTPTPNameParmError  x   2
#define DefineTPFileSpecParmError  x   4
#define DefineTPRcvAllocTimeoutParmError  x   8
#define DefineTPIncAllocTimeoutParmError  x  1
#define DefineTPTPOperationParmError  x  2

typedef struct define_tp {
   /  the fields FileSpec, RcvAllocTimeout, IncAllocTimeout are optional      /
   unsigned int     ReturnCode;              /  O-Zero if ok, else error                  /
   unsigned int     WhichParmError;          /  O-this indicates which input              /
                                                  /      parm below is in error                  /
                                                  /      uses the flags defined above          /
   unsigned char    Version;                 /  I-must be                                    /
  char                TPName[TPN_LEN];
                                                  /  I-transaction program name                /
   char                FileSpec[FILE_LEN];/  I-file specification                        /
                                                  /        use " " or null string                /
                                                  /        to indicate same as TPName /
                                                  /        do not use " " or null string      /
                                                  /        if TPName is given as a hex          /
                                                  /        string                                    /
   long        int     RcvAllocTimeout;    /   I-receive  allocate  timeout                /
                                                  /              to 65534, inclusive              /
                                                  /           -1 for default                        /
                                                  /      65535 for infinite timeout            /
                                                  /      (accept_conversation timeout in     /
                                                  /      nsd.ini for DEFINETP section)        /
   long        int     IncAllocTimeout;    /   I-incoming  allocate  timeout                /
                                                  /              to 65534, inclusive              /
                                                  /           -1 for default                        /
                                                  /      65535 for infinite timeout            /
                                                  /      (incoming_attach timeout in           /
                                                  /      nsd.ini for DEFINETP section)        /
   unsigned char    TPOperation;             /  I-Zero if program launcher started  /
                                                  /      One if operator started                /
  char                ParmString[PARM_LEN];
                                                  /  I-parameter string to be passed to  /
                                                  /      to the program if started by the  /
                                                  /      program  launcher                        /
} Define_TP;
```

NSDELTP

Use this verb to delete a TP definition after Networking Services has been started. A TP definition deleted by NSDELTP is not deleted from NSD.INI.

If this verb is issued with a TPName that has not bee defined, the verb fails, and a nonzero ReturnCode is returned.

Function Prototype

```
extern int pascal far _loadds NSDELTP(Delete_TP far );
```

Data Structure

```
#define DeleteTPVersionParmError  x   1
#define DeleteTPTPNameParmError  x   2

typedef struct delete_tp {
   unsigned int     ReturnCode;          / O-Zero if ok, else error                /
   unsigned int     WhichParmError;      / O-this indicates which input            /
                                          /      parm below is in error               /
                                          /      uses the flags defined above        /
   unsigned char    Version;             / I-must be                               /
   char               TPName[TPN_LEN];     / I-transaction program name              /
} Delete_TP;
```

NSDISCON

Use this verb to deactivate a specific link, or to deactivate all links. To determine which links are active, use the verb, NSINFO.

The force parameter, if set to nonzero, will cause the link to be deactivated even if conversations that use the link are active.

Function Prototype

```
extern int pascal far _loadds NSDISCON(Disconnect_Link far );
```

Data Structure

```
#define DisconnectLinkVersionParmError x 1
#define DisconnectLinkLinkNameParmError x 2

typedef struct disconnect_link
{
    unsigned int    ReturnCode;          / O-Zero if ok, else error          /
                                           /    5148=Network is beaconing;     /
                                           /    other - see generaldefinition  /
    unsigned int    WhichParmError;      / O-this indicates which input     /
                                           /    parm below is in error          /
                                           /    uses the flags defined above   /
    unsigned char   Version;             / I-must be                        /
    char            LinkName[LINK_LEN];/ I-Name of link to be deactivated /
                                           /    or use DISCON_ALL_LINKS to     /
                                           /    bring down all links,          /
                                           /    respecting the force parameter /
    int             Force;               / I-nonzero=force, =do not force  /
} Disconnect_Link;
```

NSINFO

Use this verb to obtain current information about:

- Attaches in the Attach list
- Links
- Local LU
- Modes
- Modes associated with partner LUs
- Partner LUs
- Sessions
- Side information definitions
- TP definitions

For some queries, provide a key field that is used to lookup the definition to be returned. The following query types support lookup:

- Links
- Modes
- Partner LUs
- Side information definitions
- TP definitions

Output Parameters

ItemsDefined	Number of items of the QueryType that are currently defined
StorageRequired	Storage, in bytes, required to hold information about all items currently defined
ItemsStored	Number of items stored at the location pointed to by QueryPointer
OutputElementSize	Size, in bytes, of a stored item

Function Prototype

```
extern int pascal far _loadds NSINFO(Query_Info far );
```

Data Structure

```
#define QueryInfoVersionParmError  x   1
#define QueryInfoStorageSizeParmError  x   2
#define QueryInfoItemsParmError  x   4
#define QueryInfoQueryPtrParmError  x   8
#define QueryInfoQueryTypeParmError  x  1
#define QueryInfoLookupKeyParmError  x  2

typedef enum {LLU,
```

```
                    ATTACH,
                     LINK,       /  supports lookup, key field is LinkName        /
                     MODE,       /  supports lookup, key field is ModeName        /
                     PLU,        /  supports lookup, key field is PLUName         /
                    PLUMODE,
                    SESSION,
                     SIDEINFO,/  supports lookup, key field is SymDestName  /
                     TPDEF       /  supports lookup, key field is TPName          /
} QUERY_ENUM;

typedef enum {lennode, endnode, networknode} NodeTypeEnum;
typedef enum {ASYNC, LAN, SDLC, TWINAX, MPTN} DLCTypeEnum;
typedef enum {PRIMARY, SECONDARY, COM3, COM4, INVALID_ADAPTER} AdapterTypeEnum;
typedef enum {SecondaryRole, PrimaryRole, NegotiableRole} StationRoleEnum;
typedef enum {AllAttaches, DefinedAttaches, NoAttaches} AttachAuthEnum;

typedef struct query_info {
   unsigned int     ReturnCode;             /  O-Zero if ok, else error                  /
   unsigned int     WhichParmError;         /  O-this indicates which input              /
                                            /     parm below is in error                 /
                                            /     uses the flags defined above          /
   unsigned int     ItemsDefined;           /  O-current number of items def'd          /
   unsigned  int    StorageRequired;    /  O-storage  required  to  hold  all          /
                                            /     elements currently defined            /
   unsigned int     ItemsStored;            /  O-number of elements stored at          /
                                            /     QueryPtr                               /
   unsigned int     OutputElementSize; /  O-size in bytes of a stored                 /
                                            /     element                                /
   unsigned char    Version;                /  I-must be   or 1 (see below)             /
   QUERY_ENUM       QueryType;              /  I-Type of query (see above)              /
   char             LookupKey[TPN_LEN];/  I-if "", then not a lookup (get          /
                                            /     all  values/size)                      /
                                            /     otherwise, search for this value  /
                                            /     (lookup).    For lookup, QueryPtr     /
                                            /     must point to a valid storage         /
                                            /     area                                   /
                                            /     if not "" and the QueryType does  /
                                            /     not support lookup, then a parm     /
                                            /     error                                  /
                                            /     Also, see special note below for  /
                                            /     mode name lookup                       /
   unsigned int     StorageSize;            /  I-Size of the QueryPtr buffer           /
                                            /     must be >   if Items >   or            /
                                            /     LookupKey is not ""                    /
   unsigned int     Items;                  /  I-Maximum number of items desired,  /
                                            /     ignored if QueryPtr = NULL             /
   void far         QueryPtr;               /  I-Pointer to array of struct (see     /
                                            /     below) whose area is allocated       /
                                            /     by the caller,                         /
                                            /     or NULL to indicate query of          /
                                            /     buffer size needed                     /
                                            /  O-pointer to array of structures       /
} Query_Info;

/  NOTE for mode name lookup:    if attempting to lookup the default mode       /
/      (8 x'4 ' chars--sometimes called the "default mode" or "blank mode       /
/      name"), the LookupKey field should be set to either "BLANK"                /
/      (optionally, with   to 3 trailing blanks), or should be a string of     /
/      length 1 or greater comprised of ASCII blanks (x'2 ').    These values  /
```

```
/     will be automatically mapped to the 8 EBCDIC blank characters              /
/     (hex 4  ) and the lookup will proceed.                                     /

/ NOTE on Version field:   the section below indicates which structures      /
/     are affected by a non-zero Version                                        /
/     1 -- Sess_Info:   enables the use of the field FQPCID, reserved1,         /
/                                 reserved2, reserved3, reserved4               /

typedef struct attach_info {
   unsigned int     SessionNumber;        / attached queued on session number    /
   char               PartnerLUName[LUN_LEN]; / partner LU name                   /
   char               ModeName[MODE_LEN];/ mode name                                  /
   char               TPName[TPN_LEN];     / transaction program name             /
} Attach_Info;

typedef struct link_info {
   / NOTE:   for DLC type MPTN and TWINAX, the RemoteLinkStation name will /
   / be the NULL string                                                          /
   char               LinkName[LINK_LEN];/ name of link                          /
   unsigned int       Sessions;             / number of sessions on this link       /
   unsigned  int      Conversations;      / number  of  conversations                /
   StationRoleEnum StationRole;          / primary or secondary link station    /
   char               DestAddr[LAN_ADDR_LEN]; / only valid for LAN               /
   char               RemoteLinkStation[LUN_LEN]; / remote link station name /
} Link_Info;

typedef struct llu_info {
   char               Version;              / version number                       /
   char               Release;              / release number                       /
   char               LUName[LUN_LEN];   / local LU name                              /
   NodeTypeEnum       NodeType;             / type of node                          /
   DLCTypeEnum        DLCType;              / type of DLC currently used            /
   AdapterTypeEnum AdapterPort;          / which adapter                        /
   char               LinkStations;       / number of link stations              /
   unsigned int       PartnerLUs;           / number of partner LUs                /
   unsigned int       Sessions;             / number of sessions                   /
   unsigned  int      Conversations;      / number  of  conversations            /
  AttachAuthEnum   AttachAuth;           /  attach  authorization                /
} LLU_Info;

typedef struct mode_info {
   char               ModeName[MODE_LEN];/ mode name                                  /
   unsigned int     MaxRU;                  / maximum RU size,   indicates          /
                                            /      LU will compute default based on /
                                            /      max link frame size              /
   unsigned int     RcvPacingWin;         / receive pacing window                /
   unsigned int     MaxNegSess;           / maximum negotiable sessions          /
   unsigned int     MinNegConWin;         / minimum negotiable conwinners        /
} Mode_Info;

typedef struct plu_info {
   char               PLUName[LUN_LEN];   / partner LU name                          /
   char               LinkName[LINK_LEN];/ assigned link                          /
   unsigned int     Sessions;             / number of sessions                   /
   unsigned int     Conversations;       / number of conversations              /
   char               ParallelSessions;   / support for parallel sessions       /
} PLU_Info;

typedef struct plum_info {
```

```
   char              PLUName[LUN_LEN];    /  partner LU name                          /
   char              ModeName[MODE_LEN];/  mode name                                    /
   unsigned int      MaxSessions;          /  maximum number of sessions               /
   unsigned int      MinConWinners;        /  minimum number of contention win         /
   unsigned int      ActiveConWinners;    /  number of active conwinners              /
   unsigned  int     ActiveConLosers;     /  number  of  active  conlosers            /
   unsigned int      Sessions;             /  number of sessions                       /
} PLUM_Info;

typedef struct sess_info {
  unsigned  int      SessionNumber;        /  identification                           /
   char              LinkName[LINK_LEN];/  assigned link                                /
   char              PLUName[LUN_LEN];    /  partner LU name                          /
   char              ModeName[MODE_LEN];/  mode name                                    /
   char              InUse;                /  =unused, conversation otherwise          /
   unsigned int      ReceiveMaxRU;         /  maximum receive RU size                  /
   char              RcvPacingWindow;     /  receive pacing window                    /
   char              SendPacingWindow;    /  send pacing window                       /
   char              Polarity;             /  =conwinner, conloser otherwise          /
   /  the following fields are available if Version == 1                              /
   unsigned int      SendMaxRU;            /  maximum send RU size                     /
   char              FQPCID[FQPCID_LEN];/  either a null string or a 16 char    /
                                           /  (plus a null char) ASCII string        /
  char               reserved1;
  char               reserved2;
  double             reserved3;
  double             reserved4;
} Sess_Info;

typedef struct side_info {
   char              SymDestName[SYMD_LEN];    /  symbolic destination name         /
   char              PartnerLUName[LUN_LEN]; /  partner LU name                      /
   char              ModeName[MODE_LEN];        /  mode name                         /
   char              TPName[TPN_LEN_OUT];       /  transaction program name          /
  ConversationSecurity
                      ConvSecurity;             /  Conversation Security type        /
   char              UserID[USERID_LEN];        /  UserID for security               /
   char              Password[PASSWORD_LEN]; /  Password for security               /
   char              Reserved1[9];              /  Reserved                          /
} Side_Info;

typedef struct tp_info {
   char              TPName[TPN_LEN];     /  transaction program name               /
   char              FileSpec[FILE_LEN];/  file specification                        /
   unsigned  int     RcvAllocTimeout;    /  receive  allocate  timeout              /
                                           /      65535 indicates infinite timeout  /
   unsigned  int     IncAllocTimeout;    /  incoming  allocate  timeout             /
                                           /      65535 indicates infinite timeout  /
   unsigned char     TPOperation;          /  Zero if program launcher started      /
                                           /      One if operator started           /
  char               ParmString[PARM_LEN];
                                           /  parameter string passed to a          /
                                           /  program-launched program              /
} TP_Info;
```

NSREJATT

Use this verb to remove a specified Attach from the Attach list. Specify which Attach to remove by specifying the number of the session on which the Attach is queued.

Function Prototype

```
extern int pascal far _loadds NSREJATT(Reject_Attach far );
```

Data Structure

```
#define RejectAttachVersionParmError  x   1

typedef struct reject_attach
{
        int                 ReturnCode;          / return code                          /
        unsigned int        WhichParmError;      / O-this indicates which input       /
                                                 /      parm below is in error         /
                                                 /      uses the flags defined above   /
        unsigned char       Version;             / I-must be                           /
        unsigned int        SessionNumber;       / I-number of session on which       /
                                                 /      attach is queued on            /
} Reject_Attach;
```

NSSTART

Use this verb to start Networking Services. Using this verb will also cause links to be activated if their link definitions are included in NSD.INI.

Function Prototype

```
extern int pascal far _loadds NSSTART(Start_NSD far );
```

Data Structure

```
#define StartNSDVersionParmError x 1

typedef struct start_nsd
{
      unsigned int     ReturnCode;          / O-Zero if ok, else error              /
      unsigned char    Links[MAX_LINKS];    / O-status of links listed in          /
                                            /     NSD.INI, non-zero=links started  /
      unsigned int     WhichParmError;      / O-this indicates which input         /
                                            /     parm below is in error           /
                                            /     uses the flags defined above     /
      unsigned char    Version;             / I-must be                            /
} Start_NSD;
```

NSSTOP

Use this verb to stop all links and stop Networking Services.

The force parameter, if set to nonzero, will cause Networking Services to be deactivated even if conversations are active.

Function Prototype

```
extern int pascal far _loadds NSSTOP(Stop_NSD far  );
```

Data Structure

```
#define StopNSDVersionParmError  x   1

typedef struct stop_nsd
{
        unsigned int     ReturnCode;          / O-Zero if ok, else error                /
        unsigned int     WhichParmError;      / O-this indicates which input           /
                                                /      parm below is in error               /
                                                /      uses the flags defined above      /
        unsigned char    Version;             / I-must be                               /
        int                Force;               / I-nonzero=force,  =do not force    /
} Stop_NSD;
```

Part 4. Appendixes, Glossary, and Index

Appendix A. Converting Networking Services/DOS Applications to Networking Services for Windows A-1

Appendix B. Getting Information from IBM's APPC Market Enablement Group B-1
APPC Developer Assistance Program (DAP) B-1
Other Services of APPC Market Enablement B-4
Get the Latest Information on CPI-C, APPC, & APPN Online B-4

© Copyright IBM Corp. 1994

Appendix A. Converting Networking Services/DOS Applications to Networking Services for Windows

If you have previously used the Networking Services/DOS product, and you are upgrading to Networking Services for Windows, note the following considerations and restrictions for converting your Networking Services/DOS application programs to Networking Services for Windows applications.

Networking Services for Windows does not support DOS CPI-C applications. A DOS CPI-C application will not run in a DOS box in Networking Services for Windows. A DOS CPI-C application must be converted into a Windows CPI-C application.

Any Networking Services/DOS application that used the node operator facility (NOF) application program interface must be relinked. For more information about the NOF API, see Chapter 6, “Using Networking Services Node Operator Application Program Interface.”

Any Networking Services/DOS application that used a the CPI-C send-receive mode of full-duplex must be recompiled. In particular, ensure that your application does not issue the Set_Send_Receive_Mode (CMSSRM) call between an Initialize_For_Incoming (CMINIC) call and an Accept_Incoming (CMACCI) call.

© Copyright IBM Corp. 1994

Converting Networking Services/DOS Applications

Appendix B. Getting Information from IBM's APPC Market Enablement Group

IBM's APPC Market Enablement group offers many types of support services to developers who use APPC, or are in the process of adding APPC support.

APPC Developer Assistance Program (DAP)

One of the support services provided by the IBM APPC Market Enablement Group is the APPC Developer Assistance Program (DAP).

The APPC DAP is open to Independent Software Vendors (ISVs) who are developing, or planning to develop, APPC or CPI-C (Common Programming Interface for Communications) support in their products. It is designed to offer all the help needed: from education about APPC, to support while your developers add APPC code to a product, through the actual marketing of the final product. The APPC DAP provides you technical, business, and marketing services related to the development and promotion of APPC support in your products.

Technical services include:

The Best of APPC, APPN and CPI-C CD-ROM

This CD contains numerous APPC and CPI-C programming samples, sample applets, white papers designed to aid with APPC/CPI-C programming and configuration topics. The CD contains the CPI-C programming reference manual, as well as thousands of lines of CPI-C sample code, in C. As a DAP member you will automatically be sent any new releases of the CD.

Technical Assistance

Technical assistance from highly experienced APPC and CPI-C programmers is provided to DAP members. This support is provided via the CompuServe APPC forum or the Internet. This technical assistance can be used by your design or programming staff to assist them in building the APPC or CPI-C components of your products. You will have a team of experts by your side.

Technical Conferences

The APPC/APPN Technical Conference (AATC) is held several times a year. As a member of the DAP, you are encouraged to attend the AATC and meet your customers. Conference schedules and fees will be sent to you prior to each conference.

APPC Developer Kits

The APPC Developer Kits contain platform-specific manuals to help your application developers design and develop APPC applications. The kits give you the ability to jump start your application development efforts by having all needed information available from the start. Kits are currently available for:

© Copyright IBM Corp. 1994

APPC Information

Communications Manager/2, AIX* SNA Services, APPC/MVS, Networking Services/DOS, APPC Networking Services for Windows, CICS/ESA*, and AS/400*.

APPC and APPN Configuration Information

Configuration information for multiple platforms that use APPC and APPN is documented in one concise document. You are free to include and ship this document with your APPC product, thus eliminating the need for you to recreate and maintain the configuration documentation. The document is available in both hard and soft copy formats.

Business services include:

Automatic Subscription to *The APPC Connection*

The APPC Connection is a free, bimonthly newsletter devoted solely to APPC, APPN and CPI-C. The current subscriber base is 33,000 readers worldwide and growing rapidly. This publication is targeted to and read by both technical and business personnel with a specific interest in APPC and APPN. Back issues of *The APPC Connection* are also available.

Opportunity to Participate in Early-Code Programs for APPC Products

As a member of the DAP, you will be notified of IBM APPC product early-code (beta) programs. Although these programs are administered by the specific product group conducting the program, you will be informed of these programs and told how to request participation.

Marketing services include:

Discounts on Advertising in *The APPC Connection*

As a member of the DAP, you are eligible for a discount on advertising space, subject to availability, in *The APPC Connection*. This discount applies solely to the first advertisement per APPC product, and is only applicable to new APPC products or announcements.

Discounts on Product Demonstration Space at the AATCs

DAP members are entitled to discounts, subject to availability of space, on product demonstration space at the AATCs.

APPC and APPN Trademarks

Logos have been developed for APPC and APPN. Use these logos when you advertise your products and on your packaging.

Your APPC Product in the APPC DAP Database

The DAP is building a database containing vendors and their products that support APPC. We are using this database to fulfill customer requests for information on products supporting APPC. Having your products listed in this database provides you with free advertising to customers looking for APPC products.

It is understood that IBM is free to change, modify or discontinue this program at any time, and to discontinue Company's membership at any time. IBM does not grant the

right of the Company to use the IBM name in conjunction with their products or advertisements.

Enrollment

To enroll in the APPC DAP or for more information, contact us at:

Internet: appcmrkt@vnet.ibm.com
CompuServe: 76711,37
IBM Mail: USIB43DB at IBMMAIL
Fax: 1-919-254-6 5
Voice Mail: 1-919-254-4957
By mail: IBM Corporation
APPC Developer Assistance Program
Dept. E42
P.O. Box 12195
Research Triangle Park, NC 277 9
USA

APPC Information

Other Services of APPC Market Enablement

Get the Latest Information on CPI-C, APPC, & APPN Online

You can find helpful CPI-C, APPC, and APPN information and utilities online via:

Internet

OS/2* BBS

CompuServe

You can get up-to-date information on CPI-C, access sample programs and tools, ask questions, and provide feedback on CPI-C and related IBM products. You can post questions about CPI-C programming and architecture, configuration, and about using APPC with any of IBM's products such as CICS, Communications Manager/2, Networking Services/DOS, APPC Networking Services for Windows, AIX SNA Server/6000, AS/400, VTAM/NCP, MVS, and VM.

Internet

A wide variety of APPC announcements, utilities, and CPI-C sample programs are available from the anonymous Internet FTP site, networking.raleigh.ibm.com in /pub/appc_appn. IBM's APPC Market Enablement team also monitors the bit.listserv.appc-l newsgroup.

OS/2 BBS

The OS/2 BBS's APPC FORUM is available on the IBM Information Network (IIN*) and the IBMLink* TalkLink facility. In the US phone 1-800-547-1283 and in Canada, 1-800-465-7999 ext. 228 to speak with a representative for more information on OS/2 BBS. Elsewhere contact your local IBM representative for more information on OS/2 BBS.

CompuServe - the APPC/APPN Forum (GO APPC)

Whether you're an experienced APPC software developer or a novice who has just started learning about distributed applications, the APPC/APPN Forum on CompuServe provides you with the information and help you need.

On the APPC/APPN Forum, you can:

Talk to the experts

Get up-to-date information

Access sample programs, tools, and utilities

Tell us what you need to use CPI-C successfully.

This forum is managed by the IBM APPC Market Enablement Team. They are committed to giving you the information and support you need to get the most from APPC.

To access the APPC/APPN Forum, type GO APPC.

APPC Information

In this forum you'll find message sections where you can ask questions and get expert guidance on:

Designing APPC Applications

Using APPC on DOS, Windows, OS/2, MVS, CICS, VM, VTAM, AIX, etc...

Using the Common Programming Interface for Communications (CPI-C)

Configuring complex APPC and APPN networks

You can also contribute to the Suggestion Box and let us know your ideas for improving APPC, CPI-C, APPN, and related products.

The APPC / APPN Forum also provides libraries, where you can get:

Answers to frequently asked questions about APPC

Success stories of businesses and governments who have used APPC and APPN to solve their client/server and networking needs

Tools and utilities like file transfer, remote execution, name server, tell, and many others

Catalogs on APPC education, development tools, platforms, and applications offered by IBM and other vendors

Recent technical papers on APPC application design and APPC products

Source code examples that illustrate classic coding techniques for APPC applications

Complete sample programs that demonstrate the power of APPC

Configuration samples that give you common definitions for building networks that support APPC applications

We invite you to contribute your own programs, samples, and information to these libraries.

Call Now for Your Free Membership: You can try CompuServe and the APPC Info Exchange Forum for yourself with no risk or obligation.

To get your free introductory membership, phone CompuServe:

In the US and Canada, phone 800-848-8199.

In the UK, phone 0800-289-378.

In Germany, phone 0130-37-32.

In the rest of Europe, phone 44-272-255-111.

Elsewhere, phone 614-457-0802.

Be sure to ask for representative 337.

You will receive:

APPC Information

A $15 introductory usage credit to explore the APPC Info Exchange Forum and other CompuServe offerings

A private ID number and password

An introductory subscription to CompuServe Magazine.

Bibliography

Softcopy

This book is also available in softcopy format for viewing with IBM* BookManager* READ or BookManager Library Readers. IBM includes the BookManager Library Readers on all IBM Online Library Softcopy Collection Kits that contain softcopy books. The library readers are limited editions of BookManager READ/2, READ/DOS, or READ for Windows that enable you to read and search softcopy books specifically enabled by IBM.

Before you use BookManager to display this book, see the license agreement on page vii. For more information on BookManager for Windows, refer to the *BookManager Library Reader for Windows: User's Guide*.

Publications

To order IBM publications, contact your IBM marketing representative or IBM authorized dealer.

Where to Find Networking Services Information

For information on Networking Services for Windows, refer to the following publications:

IBM APPC Networking Services for Windows: Getting Started, SC31-8124

IBM APPC Networking Services for Windows: Administrator's Guide, SC31-8125

IBM APPC Networking Services for Windows: Application Programmer's Reference, SC31-8126

IBM APPC Networking Services for Windows: Configuration Parameters Reference for Administrators and Application Programmers, SC31-8138.

Where to Find Related Information

Networking Services supports the Common Programming Interface for Communications (CPI-C). For more information on CPI-C, refer to either of the following:

CPI Communications Reference, SC26-4399 This book contains the CPI-C specifications as well as appendices on various IBM operating environments that support CPI-C, and an extensive bibliography of related publications for each of these environments.

CPI Communications Specification, SC31-6180 This book also contains the CPI-C specifications. It is the result of the CPI-C Implementers' Workshop, a group of both IBM and non-IBM product implementers.

For information on APPC publications, classes, platforms, and applications offered by IBM and other companies, refer to *The APPC Resource Book*, G325-0055.

For information on Systems Network Architecture (SNA) and APPC, refer to the following publications:

Program-to-Program Communication in System Application Architecture (SAA) Environments, GG24-3482

Systems Network Architecture Technical Overview, GC30-3073

SAA/LU 6.2: Distributed Networks and Applications by John J. Edmunds (McGraw-Hill, 1992, ISBN 0-07-019022-4).

For information on registering network identifiers (NETIDs), refer to *Registration Process for SNA Netids: User's Guide for SNA Network Registry*, ZZ27-7425.

For information on APPC and APPN* implementations, refer to the following publications:

APPC and CPI-C Product Implementations GG24-3520. This book shows how different products implemented LU 6.2 and compares the different product implementations with the LU 6.2 architecture.

APPN Architecture and Product Implementations, GG24-3669. This book contains a tutorial on APPN, as well as an overview of the various product implementations.

For information on APPC programming, refer to the following publication:

PC Support/400: Application Program Interface Reference, SC41-8254

© Copyright IBM Corp. 1994

Bibliography

For information on Client Access/400 for Windows 3.1, refer to the following publication:

Client Access/400 for Windows 3.1: Getting Started, SC41-3530

Client Access/400 for Windows 3.1: API and Technical Reference, SC41-3531

For information on APPC Application Suite, refer to the following publication:

APPC Application Suite Administration, SC31-6533

APPC Application Suite Programming Reference, SC31-6534

APPC Application Suite User's Guide, SC31-6532

For information on BookManager Library Reader for Windows, refer to the following publications:

BookManager Library Reader for Windows: User's Guide (softcopy only)

BookManager Library Reader for Windows: Online Reference (softcopy only)

BookManager Library Reader for Windows: Quick Reference, SC34-3067.

For information on factors that affect Networking Services performance on a LAN, refer to *AS/400 Communications: Local Area Network Guide*, SC41-0004.

For information on synchronous data link control (SDLC), refer to *Synchronous Data Link Control Concepts*, GA27-3093.

For information on International Standards Organization (ISO) standards, contact:

American National Standards Institute
1430 Broadway
New York, NY 10018

Configuration Information for Other Platforms: For detailed information about configuring platforms other than Networking Services, refer to the following publications:

AIX SNA Services/6000

- *AIX SNA Services/6000 Reference*, SC31-7014
- *Using AIX SNA Services/6000*, SC31-7002

APPC/MVS

- *MVS/ESA Planning: APPC Management*, GC28-1110

APPC/VM

- *VM/ESA Connectivity Planning, Administration, and Operation*, SC24-5448

AS/400

- *AS/400 Communications APPC Guide*, SC41-8189
- *AS/400 Communications APPN Guide*, SC41-8188
- *AS/400 Programming: Control Language Reference*, SC41-0030
- *AS/400 Programming Reference Summary*, SC21-8076

CICS/ESA

- *CICS/ESA: Resource Definition (Online)*, SC33-0666
- *CICS/ESA: System Definition Guide*, SC33-0664

OS/2

- *Communications Manager/2 APPC Programming Guide and Reference*, SC31-6160
- *Communications Manager/2 Configuration Guide*, SC31-6171
- *Communications Manager/2 Installation, Configuration and Administration*, SBOF-4386

The paper, *IBM Multi-Platform APPC Configuration Guide*, shows how to configure APPC and APPN on various platforms, and shows which values in one computer's configuration must match values in a partner computer's configuration. This paper is available through CompuServe (described below) as the following files:

MPCONP.ZIP PostScript version

MPCONT.ZIP ASCII text version

MPCONF.ZIP Sample files

If you do not have access to CompuServe, ask your IBM representative to get the MPCONFIG PACKAGE from the MKTTOOLS disk.

Glossary

This glossary defines terms and abbreviations used in this book. If you do not find the term you are looking for, refer to the index or to the *IBM Dictionary of Computing*, SC20-1699.

A

accept. One of two CPI-C calls, Accept_Conversation (CMACCP) or Accept_Incoming (CMACCI), that is issued by a local program to begin a conversation requested by a remote computer sending an Attach.

Advanced Program-to-Program Communication (APPC). (1) The general facility characterizing the LU 6.2 architecture and its various implementations in products. (2) Sometimes used to refer to the LU 6.2 architecture and its product implementations as a whole, or an LU 6.2 product feature in particular, such as an *APPC application program interface*.

alias. (1) An alternative name used to identify an object, a logical unit, or a database. (2) A nickname set up by the network administrator for a file, printer, or serial device.

allocate. (1) To assign a resource to perform a specific task. (2) In APPC, a type of verb used to assign a session to a conversation for its use. Contrast with *deallocate*.

American National Standard Code for Information Interchange (ASCII). The standard code with a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems and data communication systems. The ASCII set consists of control characters and graphic characters.

API. Application program interface.

APPC. Advanced Program-to-Program Communication.

application. A program or set of programs that perform a task.

application program interface (API). A formally-defined programming language interface between a system control program or a licensed program and the user of a program.

application transaction program. A program written for or by a user to process the user's application; in an SNA network, an end user of a type 6.2 logical unit. Contrast with *service transaction program*.

ASCII. American National Standard Code for Information Interchange.

asynchronous. Data transmission in which transmission of a character or block of characters can begin at any time but in which the bits that represent the character or block have equal time duration. Contrast with *synchronous transmission*

Attach. In SNA, the request unit that flows on a session to initiate a conversation.

Attach Manager. The component of APPC that matches Attaches received from remote computers to accepts issued by local programs.

Attach list. The list of received Attaches currently eligible to be accepted by any program that issues an accept.

B

basic conversation. In APPC, a conversation between two transaction programs using the APPC basic conversation API. Programs using the basic conversation API exchange data records in an SNA-defined format. This format is a stream of data containing 2-byte length prefixes that specify the amount of data to follow before the next prefix. Typically, service transaction programs use basic conversations and end-user transaction programs use mapped conversations. However, either type of program may use either type of conversation. Contrast with *mapped conversation*.

BIND. In SNA, the response/request unit (RU) involved in activating a session between logical units (LUs) or control points (CPs).

block. (1) A string of records, a string of words, or a character string formed for technical or logical reasons to be treated as an entity. (2) A set of things, such as words, characters, or digits handled as a unit. (3) To wait, usually for an input/output (I/O) event to complete or for a resource to become available.

© Copyright IBM Corp. 1994

blocking. A CPI Communications call-processing mode in which a call operation completes before control is returned to the program. The program is blocked (unable to perform any other work) until the call operation is completed.

block length. The number of records, words, or characters in a block.

block size. Synonym for *block length*.

buffer. (1) A routine or storage used to compensate for a difference in rate or flow of data, or time or occurrence of events, when transferring data from one device to another. (2) An area of storage that is temporarily reserved for use in performing an input/output operation, into which data is read or from which data is written.

buffer pool. The shared area in memory that contains communication buffers for all Networking Services conversations to or from the workstation. This area is set aside when Networking Services is loaded, and released when Networking Services is unloaded.

C

call. (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (2) In data communication, the actions necessary to make a connection between two stations on a switched line. (3) In communications, a conversation between two users. (4) To transfer control to a procedure, program, routine, or subroutine. (5) To attempt to contact a user, regardless of whether the attempt is successful.

CICS. Customer Information Control System.

CICS/VS. Customer Information Control System for Virtual Storage.

class of service (COS). A designation of the transport network characteristics, such as route security, transmission priority, and bandwidth, needed for a particular session. The class of service is derived from a *mode name* specified by the initiator of a session.

class of service database. A database maintained independently by each network node and optionally by APPN end nodes. It contains one entry per class of service name; each database entry contains:

A definition of the acceptable values for transmission group (TG) and node characteristics for routes described by that class of service name and the weight function to be used to compute the weights of nodes and TGs that meet the acceptable values.

The transmission priority to be used for traffic that flows on routes described by that class of service name.

client. (1) A user. (2) A functional unit that receives shared services from a server.

CNOS. (1) Change Number of Sessions. (2) A service transaction program which changes the maximum number of sessions allowed between a pair of LUs from 0 to a positive number, or from a positive number to O. One or more of the sessions can be automatically activated.

Common Programming Interface for Communication (CPI-C). A consistent application programming interface for applications that require program-to-program communication. The interface makes use of APPC to create a rich set of interprogram services, including:

Sending and receiving data

Synchronizing processing between programs

Notifying a partner of errors in the communication.

communication. The transmission and reception of data. Synonymous with *data communication*.

communication adapter. A circuit card with associated software that enables a processor, controller, or other device to be connected to a network.

communications port. (1) An access point for data entry or exit to or from a communication device such as a terminal. (2) On a personal computer, a synchronous or asynchronous serial port to which a modem can be attached. Synonymous with *port*.

communications line. The physical link, such as wire or a telephone circuit, that connects one or more workstations to a communications control unit or that connects one control unit to another. See also *data link*.

Communications Manager. A component of OS/2 that lets a workstation connect to a remote computer and use the resources of that computer as well as the resources of other personal computers to which the workstation is attached. Communications Manager provides APIs so that users can develop their own applications.

compiler. (1) A program that translates a source program into an executable program (an object program). (2) A program that decodes instructions written as pseudo codes and produces a machine language program to be executed at a later time. (3) A program that translates instructions written in a high-level programming language into machine language.

confirmation. (1) Acknowledgement by a program that data has been received or a request completed. (2) A prompt in a menu window wherein a program questions the user when the consequences of a user action are significant.

conloser. Contention loser.

conwinner. Contention winner.

contention. In a session, a situation in which both NAUs attempt to initiate the same action at the same time, such as when both attempt to send data in a half-duplex protocol (half-duplex contention), or both attempt to start a bracket (bracket contention). At session initiation, one NAU is defined to be the contention winner; its action will take precedence when contention occurs. The contention loser must get explicit or implicit permission from the contention winner to begin its action.

contention loser. In APPC, the LU that must request and receive permission from the session partner LU to allocate a session. Contrast with *contention winner*.

contention-loser polarity. In APPC, the designation that an LU is the contention loser for a session. Contrast with *contention-winner polarity*.

contention-loser session. To a NAU, a session for which it was defined during session initiation to be the contention loser.

contention winner. The LU that can allocate a session without requesting permission from the session partner LU. Contrast with *contention loser*.

contention-winner polarity. In APPC, the designation that an LU is the contention winner for a session. Contrast with *contention-loser polarity*.

contention-winner session. To a NAU, a session for which it was defined during session initiation to be the contention winner.

control block. (1) A storage area used by a computer program to hold control information. (2) A specifically formatted block of information provided from the application program to an API to request an operation.

conversation. A logical connection between two transaction programs using an APPC session.

conversation characteristics. A set of characteristics for each conversation used by a program that is maintained by CPI-C. These characteristics are established for each program on a per-conversation basis. The initial values assigned to the characteristics depend on the program's role in starting the conversation.

conversation ID. A unique ID for each conversation between two transaction programs.

conversation security. In APPC, a process that allows validation of a user ID or group ID and password before establishing a connection.

conversation state. The condition of a conversation that reflects what the past action on that conversation has been and that determines what the next set of actions may be.

conversation type. In APPC, a conversation can be either a basic conversation or a mapped conversation.

conversion. The process of changing from one form of representation to another; for example, to change from decimal representation to binary representation.

COS. Class of service.

CPI Communications. Common Programming Interface for Communications.

CPI-C. Common Programming Interface for Communications.

CSD. Corrective Service Diskette.

Customer Information Control System (CICS). An IBM-licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining data bases.

Glossary

D

dangling conversation. Conversations that are left if the application program terminates, normally or abnormally, without deallocating them with the deallocate (CMDEAL) call of CPI-C. Your workstation still has a conversation connection to the remote computer, but it is no longer usable. Networking Services deallocates dangling conversations.

data buffer. blocks of memory used to send or receive data.

data communication. The transmission and reception of data.

deadlock. (1) Unresolved contention for the use of a resource. (2) In APPC, an error condition in which processing cannot continue because each of two elements of the process is waiting for an action by or a response from the other.

deallocate. To release a resource that is assigned to a specific task. Contrast with *allocate*.

default. An alternative value, attribute, or option that is assumed when none has been specified.

DEFINETP.NSD. The Networking Services/DOS configuration file that contains transaction program definitions. See TP.

distributed application. (1) An application for which the component application programs are distributed between two or more interconnected processors. (2) In SAA usage, an application that is designed to have its parts split across and run on two or more linked systems.

DLL. Dynamic link library.

duplex. Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full-duplex*. Contrast with *half-duplex*.

dynamic link library (DLL). A program or routine that can be loaded by an application or as part of a program.

E

EBCDIC. Extended binary-coded decimal interchange code.

end user. The ultimate source or destination of application data flowing through an SNA network. An end user can be an application program or a workstation operator.

extended binary-coded decimal interchange code (EBCDIC). A coded character set consisting of 8-bit coded characters used by mainframe computers.

F

FID. Format identification field.

field. (1) In a record, a specified area used for a particular category of data, for example a group of card columns in which a wage rate is recorded. (2) In a data base, the smallest unit of data to which you can refer. A field contains one unit of information from a column within a row of data in a table.

file. A set of related records treated as a unit.

file name. (1) The name assigned to a set of input data or output data. (2) The portion of the identifying name that precedes the extension.

filespec. The name and location of a file. The format is dependent on the storage medium of a file. For example, c:\path\file_name.ext.

flush. A program uses the Flush (CMFLUS) call to empty the local LU's send buffer for a given conversation. When notified by CPI Communications that a Flush has been issued, the LU sends any information it has buffered to the remote LU. The information that can be buffered comes from the Allocate, Send_Data, or Send_Error call.

flushing. In APPC, the process of sending through the network all remaining buffered data generated by a transaction program. The transaction program issues the flush verb to begin the process. It also occurs if the network operator issues the command.

FMD. Function management data.

FMH. Function management header.

FM header. Function management header.

full-duplex. Synonym for *duplex*. Contrast with *half-duplex*.

function management data (FMD). An RU category used for end-user data exchanged between logical units (LUs) and for requests and responses exchanged between network services components of LUs, PUs, and control points.

function management header (FMH). One or more headers, optionally present in the leading request units (RUs) of an RU chain that allow one half-session to: (a) select a destination at the session partner and control the way in which the end-user data it sends is handled at the destination, (b) change the destination or the characteristics of the data during the session, and (c) transmit between session partners status or user information about the destination, such as a program or device). Function management headers can be used with LU type 1, 4, and 6.2 protocols.

H

half-duplex. In two-way communication, where only one user transmits at a time. Contrast with *full-duplex*.

hang. A situation where a program is waiting indefinitely for some condition to be satisfied. Contrast with *deadlock*.

hex. Hexadecimal.

hexadecimal (hex). A numbering base where 4 bits are used to represent each digit. The digits can have one of 16 values, usually numbered 0,1,2,..., 9,A,B,C,D,E,F.

I

I frame. Information frame.

inactive program. A program that has been started but is suspended so that it is no longer running.

incoming Attach. An Attach which is received by Networking Services.

independent LU. An LU whose operation does not depend on an SNA host system.

information (I) frame. A frame in I format used for numbered information transfer.

input field. An unprotected field on a display surface in which data can be entered, modified, or erased.

inquiry transaction. A transaction that does not update a database.

integer. One of the numbers 0, +1, -1, +2, -2....

interactive program. A program that is running (active) and ready to receive (or is currently receiving) input from the user. See also *active program*. Contrast with *non-interactive program*.

interrupt. A suspension of a process, such as execution of a computer program caused by an external event, and performed in such a way that the process can be resumed. Programs in the workstation can register themselves to be called by other programs by placing their address at an interrupt level.

K

KB. Kilobyte.

keyword. One of the predefined words of a computer, command language, or an application.

kilobyte. 1024 bytes.

L

LAN. Local area network.

latency. The time interval between the instant at which an instruction control unit initiates a call for data and the instant at which the actual transfer of the data starts.

length field. In a basic conversation, the 2-byte length prefix containing the logical records exchanged by transaction programs.

linkage editor. A computer program for creating load modules from one or more object modules or load modules by resolving cross-references among the modules, and if necessary, adjusting addresses. Synonymous with *linker.*

linker. Synonym for *linkage editor*.

LL field. See *length field*.

local. Residing on the user's node or workstation.

local transaction program. In APPC, the transaction program at the local end of the conversation. Contrast with *remote transaction program*.

logical record. (1) A record independent of its physical environment. Portions of the same logical record may be located in different physical records, or several logical records or parts of logical records may be located in one physical record. (2) A record from the standpoint of its content, function, and use rather than its physical attributes; that is, one that is defined in terms of the information it contains.

logical unit (LU). A port providing formatting, state synchronization, and other high-level services through which an end user communicates with another end user over an SNA network.

logical unit 6.2 (LU 6.2). A particular type of SNA LU that provides a connection between resources and transactions programs running on different network nodes. These logical units interact with each other as peers. See *APPC*.

LU. Logical unit.

LU 6.2. A type of logical unit that supports general communication between programs in a distributed processing environment. LU 6.2 is characterized by (a) a peer relationship between session partners, (b) efficient utilization of a session for multiple transactions, (c) comprehensive end-to-end error processing, and (d) a generic application program interface consisting of structured verbs that are mapped into a product implementation.

LU-LU session. In SNA, a session between two logical units (LUs) in an SNA network. It provides communication between two end users or between an end user and an LU services component.

LU type. The classification of an LU in terms of the specific subset of SNA protocols and options it supports for a given session, specifically:

The mandatory and optional values allowed in the session activation request.

The usage of data stream controls, function management headers (FMHs), request unit parameters, and sense data values.

Presentation services protocols such as those associated with FMH usage.

M

MB. Megabyte.

mapped conversation. In APPC, a conversation between two transaction programs using the APPC mapped conversation API. Programs using this API exchange data records with arbitrary data formats agreed upon by the applications' programmers. Typically, end-user transaction programs use mapped conversation and service transaction programs use basic conversations. However, either type of program may use either type of conversation. Contrast with basic conversation.

megabyte. 1 048 576 bytes.

message log. In Networking Services, an ASCII text file named MSGLOG.NSD. This file contains error messages and informational messages logged by Networking Services.

mode. See *mode name*.

mode definitions table. The Networking Services table that contains mode definitions. It is initialized when Networking Services is started by the contents of the MODE.NSD file (if it exists), and can be altered by node operator API verbs. See also *mode name*.

mode name. In APPC, a name that a program uses to request a specific set of network connectivities of a session the program wants to use for a conversation. These connectivities include, for example, the highest synchronization level for conversations on the sessions, the class of service for the sessions, and the session routing and delay characteristics. See also *class of service*.

MODE.NSD. The Networking Services/DOS configuration file that initializes the mode definitions table when Networking Services/DOS is started. See also *mode name*.

multitasking. (1) Pertaining to the concurrent execution of two or more tasks by a computer. (2) Multiprogramming that provides for the concurrent performance, or interleaved execution, of two or more tasks.

N

negotiable BIND. In SNA, a capability that allows two LU-LU half-sessions to negotiate the parameters of a session when the session is being activated.

negotiable link station. The capability of a link station to assume either a primary link-station or secondary link-station role and to negotiate with a partner link station during link activation which role it will assume.

network-qualified name. A 1- to 17-byte name consisting of an optional qualifier concatenated to a 1- to 8-byte type-1134 symbol-string name; when present, the qualifier contains a 1- to 8-byte type-1134 symbol-string network identifier concatenated with a period (when the qualifier is not present, the period is omitted).

node operator application program interface. An application program interface (API) offered by Networking Services that allows an application program to enter commands and provide configuration information after Networking Services has been started. This API is also utilized to provide a Windows user interface.

non-blocking. A CPI Communications call-processing mode in which a call operation completes immediately, if possible. If the call operation cannot complete immediately, control is returned to the program with the CM_OPERATION_INCOMPLETE return code. The call operation remains in progress, and completion of the call operation occurs at a later time. Meanwhile, the program is free to perform other work.

noninteractive program. An active program that is running but cannot receive input from the keyboard or other interactive input device.

normal termination. The state of a program that has ended successfully. Contrast with *abnormal termination*.

NSD.INI. The Networking Services file that is used to initialize characteristics for a communications adapter, links to other computers, and mode, side information, and transaction program definitions.

NSDR.OVL. The nonresident component for DOS programs.

NSDW.DLL. The Networking Services for Windows component that provides the CPI-C application program interface.

O

object code. The output from a compiler or assembler, which is itself executable machine code or is suitable for processing to produce executable machine code.

object file. The machine-level program produced as the output of an assembly or compiled operation. In DOS and OS/2, these files generally have the file extension .OBJ. Synonymous with *object module*.

object module. Synonym for *object file*.

offset. The number of measuring units from an arbitrary starting point in a record, area, control block, or a segment to some other point.

one-way bracket. A bracket in which data is sent from one NAU to another in a single chain with begin bracket, conditional end bracket, and exception response requested. When one-way brackets are used on CP-CP sessions, they are always sent on the contention-winner session.

one-way conversation. A conversation in which data is sent from one transaction program (the source) to another (the target) with no response requested and that is released after the data is sent. If the source transaction program terminates as soon as it releases the conversation, the data may still be in transit; thus, the source and target TPs are not necessarily active at the same time.

OOID. Outstanding operation identifier.

P

parallel session. In SNA, two or more concurrently active sessions between the same two LUs using different pairs of network addresses. Each session can have independent session parameters.

parameter. (1) A variable that is given a constant value for a specified application and that may denote the application. (2) An item for which the operator specifies a value or for which the system provides a value. (3) A name in a procedure that is used to refer to an argument passed to that procedure.

parity. The determination whether the number of ones (or zeros) in an array of binary digits is odd or even.

partner logical unit (PLU). In Systems Network Architecture (SNA), the remote participant in a session.

partner transaction program. The transaction program engaged in an APPC conversation with a local transaction program.

password. (1) A unique string of characters that a program, computer operator, or user must supply to meet security requirements before gaining access to data. (2) A code or signal, usually confidential, that enables its user to have full or limited access to a station.

path. (1) The route used to locate files on a disk or diskette, consisting of a drive and directories. (2) In the IBM Token-Ring Network, a route between any two nodes. (3) In a data base, a sequence of segment occurrences from the root segment to an individual segment. (4) In SNA, the series of path control network components (path control and data link control) that are traversed by the information exchanged between two NAUs. A path consists of a virtual route and its route extension, if any.

peer-to-peer. The communication between two SNA LUs that is not managed by a host. Commonly used when referring to LU 6.2 nodes.

PLU. (1) Primary logical unit. (2) Partner logical unit.

positional parameter. A parameter that must be positioned in a specified location relative to other parameters.

primary half-session. In SNA, the half-session that sends the session activation request. Contrast with *secondary half-session*.

procedure. (1) The course of action taken for the solution of a problem. (2) The description of the course of action taken for the solution of a problem. (3) A sequenced set of statements that may be used at one or more points in one or more computer programs, and that usually has one or more input parameters and yields one or more output parameters. (4) The actions taken to implement protocols. (5) A named collection of related statements, and possibly control statements, that perform a particular task.

procedure call. In programming languages, a language construct for invoking execution of a procedure.

process. (1) A systematic sequence of operations to produce a specified result. (2) A unique, finite course of events defined by its purpose or by its effect, achieved under given conditions. (3) An executing function or a function that is waiting to be executed. (4) To perform operations on data in a process.

protocol. (1) A specification for the format and relative timing of information exchanged between communicating parties. (2) The set of rules governing the operation of functional units of a communication system that must be followed if communication is to be achieved. (3) In SNA, the meanings of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

pseudonym file. A file, provided as part of a CPI-C platform, which provides readable names for programmers to use when developing CPI-C

R

receive. To obtain a message or file from another computer. Contrast with *send*.

record. A set of data treated as a unit. Records are often fixed-length fields, particularly in database transactions.

record format. The definition of how data is structured in the records contained in a file. The definition includes record names, field names, and field descriptions, such as length and data type.

remote. (1) Synonym for *link-attached*. (2) Pertaining to a system, program, or device that is accessed through a telecommunication line. (3) In APPC, indicates the partner LU or transaction program is not at the local node.

remote transaction program. In APPC, the transaction program at the remote end of the conversation. Contrast with *local transaction program.*

request header (RH). Control information preceding a request unit. See also *request/response header*.

request/response header (RH). Control information, preceding a request/response unit (RU), that specifies the type of RU (request unit or response unit) and contains control information associated with that RU.

request/response unit (RU). A generic term for a request unit or a response unit. See also *request unit* and *response unit*.

request unit (RU). A message unit that contains control information, end-user data, or both.

reset. To return a device or process to a clear or initial state.

resident. Pertaining to computer programs or data while they remain on a particular storage device.

resident program. A program that remains in a particular area of storage.

response header (RH). A header, optionally followed by a response unit, that indicates whether the response is positive or negative and that may contain a pacing response.

response unit (RU). A message unit that acknowledges a request unit; it may contain prefix information received in a request unit. If positive, the response unit may contain additional information such as session parameters in response to BIND, or if negative, contains sense data defining the exception condition.

return code. The portion of all CPI-C calls which provides information about the success or failure of the call.

RH. (1) Request header. (2) Response header.

RU. (1) Request unit. (2) Response unit.

RU buffer pool. See *buffer pool*.

S

secondary half-session. In SNA, the half-session that receives the session-activation request. Contrast with *primary half-session*.

secondary logical unit (SLU). In Systems Network Architecture (SNA), the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. A logical unit can contain secondary and primary half-sessions for different active LU-LU sessions. See *primary logical unit (PLU)*. See also *secondary half-session*

send. To send a message or file to another computer. Contrast with *receive*.

sense data. In SNA, a set of data sent with a negative response, indicating the reason for the response.

server. (1) On a local area network (LAN), a data station that provides facilities to other data stations. (2) A computer that shares its resources with other computers on the network.

service transaction program. Any IBM-supplied transaction program running in a network accessible unit. Contrast with *application transaction program*.

session. (1) A connection between two stations or NAUs that allows them to communicate. (2) The period of time during which a user of a terminal can communicate with an interactive system; usually, the elapsed time between logon and logoff. (3) In SNA, a logical connection between two NAUs that can be activated, tailored to provide various protocols, and deactivated, as requested. The session activation request and response can determine options for such things as the rate and concurrency of data exchange, the control of contention and error recovery, and the characteristics of the data stream. Sessions compete for network resources such as the links within the patch control network. For routing purposes, each session is identified by the network or local addresses of the session partners. See also *CP-CP session*, *half-session*, and *LU-LU session*.

session activation. In SNA, the process of exchanging a session activation request and a positive response between NAUs. Contrast with *session deactivation*.

session deactivation. In SNA, the process of exchanging a session deactivation request and response between NAUs. Contrast with *session activation*.

session limit. In SNA, the maximum number of concurrently active LU-LU sessions a particular LU can support.

side information. System-defined values required by CPI-C to initialize a conversation. These values include the partner LU name, mode name, and transaction program name. In Networking Services, this information appears in the side information table.

side information table. The Networking Services table that contains side information. See *side information*.

SIDEINFO.NSD. The Networking Services/DOS configuration file that initializes the side information table when you start Networking Services/DOS. See *side information*.

single session. In SNA, a session that is the only session connecting two LUs. Contrast with *parallel session*.

SLU. secondary logical unit.

SNA. Systems Network Architecture.

SNA network. The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network accessible units; boundary function, gateway function, and intermediate session routing function components; and the transport network. An SNA network can use subarea protocols (see Subarea Network) or APPN protocols (see APPN) or both.

SNASVCMG mode name. SNA service manager mode name. This is the architecturally defined mode name identifying sessions on which CNOS is exchanged.

SNASVCMG session. A session that uses the SNASVCMG mode name.

source code. Synonym for *source program*.

source program. A computer program expressed in a source language.

stack. An area in memory that stores temporary register information, parameters, and return addresses of subroutines.

starter-set call. A call that allows for simple communication of data between two programs, and assumes the program uses the initial values for CPI-C conversation characteristics.

subfield. A subdivision of a field, that is, a field within a field.

subroutine. A sequence of statements that can be used in one or more computer programs and at one or more points in a computer program.

switch. An action that moves the input focus from one area to another. A switch can be within the same window or from one window to another. See also *toggle.*

symbolic destination name. A field in the side information table. CPI-C uses the symbolic destination name to access information required to initialize a conversation.

synchronization level. In APPC, the specification indicating whether the corresponding transaction programs exchange confirmation requests and replies.

system. An assembly of components united by some form of regulated interaction to form an organized whole.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through the networks and controlling the configuration and operation of networks.

The purpose of the layered structure of SNA is to allow the ultimate origins and destinations of information to be independent of, and unaffected by, the way in which the specific SNA network services and facilities used for information exchange are provided.

T

terminal. (1) A point in a system or network at which data can either enter or leave. (2) A device, usually equipped with a keyboard and display, capable of sending and receiving information over a link.

terminate. In SNA, a request unit that is sent by a logical unit (LU) to its system services control point (SSCP) to cause the SSCP to start a procedure to end one or more designated LU-LU sessions.

timeout condition. A condition indicating that a specified amount of waiting time has elapsed without the occurrence of an expected event.

TP. Transaction program.

TP name. Transaction program name. The TP_name specifies the name of the remote program and comes from the APPC architecture, where programs are referred to as transaction programs.

trace. (1) A record of data that provides a history of events that occurred in a system. (2) The process of recording the sequence in which the statements in a program are executed and, optionally, are the values of the program variables used in the statements.

transaction. An exchange between a workstation and a program, between two workstations, or between two programs that accomplishes a particular action or result.

transaction processing. A sequence of operations on a database that is viewed by the user as a single, individual operation.

transaction program (TP). A program that uses the APPC application programming interface (API) to communicate with a partner application program in the same node or at a partner node. A transaction program has a valid transaction program identifier (TP ID).

U

UNBIND. In SNA, to deactivate a session between logical units (LUs). Contrast with *BIND*.

upper memory. An area of memory consisting of 384KB of space in PC DOS computers and it is immediately adjacent to the 640KB of conventional memory. Upper memory is not considered part of the total computer memory because programs cannot store information in this area. This area is normally reserved for running the system's hardware, such as the monitor. DOS 5 allows enabled programs such as Networking Services to access portions of upper memory.

user id. A unique name that identifies a user to the network.

user session. Any session other than a SNASVCMG session.

V

verb. A reserved word that expresses an action to be taken by an application programming interface (API), a compiler, or an object program.

Virtual Telecommunications Access Method (VTAM). A set of programs that control communication between nodes and application programs running on a host system.

VTAM. Virtual Telecommunications Access Method.

W

wait condition. The normal situation in which a transaction program waits indefinitely for its partner program to take some action. Contrast with *deadlock*.

Windows user interface. A Windows end user interface that utilizes the node operator application program interface to allow an end user to enter command and configuration parameters through Windows dialog boxes.

workstation. A terminal or personal computer, usually one that is connected to a mainframe or within a network, at which a user can run applications. See also *system*.

X

X/Open. A standards group charged with creating a standard interface between software packages and UNIX**.

Glossary

Index

A

Accept_Conversation (CMACCP) 5-1, 5-8, 5-9
  See also accepting a conversation
Accept_Incoming (CMACCI) 4-3, 5-1, 5-8, 5-11
  See also accepting a conversation
accepting a conversation
  deviations from CPI-C architecture 2-8
  processing accepts 5-8
  waiting 4-3
activating links 6-11, 6-13, 6-14, 6-16
activating sessions 6-6
Allocate (CMALLC)
  implementing a conversation 3-15
Allocate return codes 6-8
API
  See application programming interface (API)
APPC and CPI-C Product Implementations X-1
application models
  choosing 3-1
  client/server 3-2
  distributed 3-3
  transaction processing 3-3
application programming interface (API)
  EHNAPPC API 1-1
  node operator 6-1—35
application subsystem verbs
  See ?
APPN Architecture and Product Implementations X-1
ASCII
  defined X-3
Attach
  authorization 5-6, 6-10
  incoming 3-15, 3-20, 5-1
Attach list 5-2
Attach Manager 3-12, 5-2
authorization, of Attaches 6-10
automatically starting a program 3-12

B

Batch Send transaction 3-10
  See also file-transfer transaction
BLANK (mode name) 2-8, 6-6
BookManager READ
  documentation X-2
BookManager READ *(continued)*
  license agreement for xi, X-1

C

calling conventions
  C 2-7
  PASCAL 2-6, 2-7
calls, CPI-C
  conventions 2-6
  supported by Networking Services 2-4
cascading server program 3-15
CICS
  See Customer Information Control System (CICS)
client program 3-15
client/server application model 3-2
CM_PRODUCT_SPECIFIC_ERROR return code 2-9
CM_PROGRAM_PARAMETER_CHECK 2-11
CMACCI (Accept_Incoming) 4-3, 5-1, 5-8, 5-11
  See also accepting a conversation
CMACCP (Accept_Conversation) 5-1, 5-8, 5-9
  See also accepting a conversation
CMALLC (Allocate)
  implementing a conversation 3-15
CMC.H
  See pseudonym files, in C
CMCFM
  See Confirm (CMCFM)
CMCFMD
  See Confirmed (CMCFMD)
CMDEAL (Deallocate) 2-9
CMSERR (Send_Error) 3-9, 3-16, 3-17
CMSLD (Set_Log_Data) 2-9
CMSMN (Set_Mode_Name) 2-8
CMSTPN (Set_TP_Name) 5-11
CMWCMP (Wait_For_Completion) 5-11
Common Programming Interface for Communications (CPI-C)
  deviations from CPI-C architecture 2-11
  function calls
    for other languages 2-7
    in C 2-6
CompuServe B-1
Confirm (CMCFM) 3-17, 3-18, 4-4
confirmation
  accepting 3-17

© Copyright IBM Corp. 1994

confirmation *(continued)*
rejecting 3-17
requesting 3-17
Confirmed (CMCFMD) 3-17
confirmed delivery transaction
See credit-check transaction
conventions used in this book xii
conversation
accepting 3-15, 5-8
allocating 3-19, 3-20
basic 3-19
choosing a conversation type 3-1, 3-12, 3-13
dangling 2-9
deallocating
cause of excess system processing 3-20
dangling conversations 2-9
one side of a conversation 3-19
implementing 3-1, 3-14
incoming 3-15
mapped 3-19
sending data 3-15
conversational reply transaction
See database-update transaction
CPI Communications Reference 2-1, X-1
CPI Communications Specification 2-1, X-1
CPI-C calls
conventions 2-6
supported by Networking Services 2-4
CPSVCMG mode name 2-8
credit-check transaction 3-8, 3-10, 3-11
Customer Information Control System (CICS) 3-3

D

dangling conversation 2-9
data buffer size 3-19
data buffers
basic conversation 3-12
mapped conversation 3-12
database-update transaction 3-11
deadlock conditions
avoiding 4-1—4-6
identifying 4-5
stopping 4-5, 4-6
understanding 4-2
Deallocate (CMDEAL) 2-9
deallocating conversations
cause of excess system processing 3-20
dangling conversations 2-9
one side of a conversation 3-19
diagnosing errors 2-9
directory
LIB 2-7
distributed application model 3-3

E

errors, reporting 3-16
extracting conversation information 3-17, 3-19

F

file-transfer transaction 3-10
flush call 3-18, 3-19
forcing data to be transmitted 3-15
full-duplex conversation 3-4, 3-6, 3-16, 3-18
error reporting 3-17
wait conditions 4-2

H

half-duplex flip-flop conversation 3-6, 3-16, 3-18
confirmation 3-17, 3-18
deallocating 3-19
error reporting 3-17
wait conditions 4-2

I

IBM APPC Networking Services for Windows: Application Programmer's Reference
conventions used in xii
prerequisites for using xi
terms used in xii
IBM C/2 2-6
IBM Library Reader xi
IBM Multi-Platform APPC Configuration Guide X-2
IMS
See Information Management System (IMS)
INCLUDE subdirectory 6-3
incoming Attach 5-1
processing 5-6
incoming conversation 3-15
Information Management System (IMS) 3-3
inquiry transaction 3-7, 3-11
International Standards Organization (ISO) X-2

L

LIB subdirectory 2-7
Library Reader
  See IBM Library Reader
link
  activating 6-11, 6-13, 6-14, 6-16
linking with CPI-C libraries 2-7
LL (length field) 3-12, 3-19
local TP name 5-11

M

Microsoft C 2-6
Microsoft Windows
  using with Networking Services 2-11
mode
  blank 2-8, 6-6
  mode name 2-8
mode name
  CPSVCMG 2-8
  SNASVCMG 2-8
models, application
  choosing 3-1
  client/server 3-2
  distributed 3-3
  transaction processing 3-3
multiple accepts 2-8, 5-12
multiple conversations 3-4

N

Networking Services
  deviations from CPI-C 2-11
node operator application program interface
  using 6-1—35
  verbs
    NSACT 6-6
    NSATTACH 5-6, 6-10
    NSCONLAN 6-11
    NSCONMPT 6-13
    NSCONREM 6-14
    NSCONTWX 6-16
    NSDEACT 6-17
    NSDEASES 6-18
    NSDEFMO 6-19
    NSDEFSI 6-21
    NSDEFTP 6-24
    NSDELMO 6-20
    NSDELSI 6-23
    NSDELTP 6-26
node operator application program interface *(continued)*
  verbs *(continued)*
    NSDISCON 6-27
    NSINFO 6-28
    NSREJATT 6-32
    NSSTART 6-33
    NSSTOP 6-34
node operator facility (NOF) 6-2
  return values and fields 6-2
non-blocking 3-18, 5-11
NSACT verb 6-6
NSATTACH verb 5-6, 6-10
NSCONLAN verb 6-11
NSCONMPT 6-13
NSCONREM verb 6-14
NSCONTWX 6-16
NSDEACT verb 6-17
NSDEASES verb 6-18
NSDEFMO verb 6-19
NSDEFSI verb 6-21
NSDEFTP verb 6-24
NSDELMO verb 6-20
NSDELSI verb 6-23
NSDELTP verb 6-26
NSDISCON verb 6-27
NSINFO verb 6-28
NSREJATT verb 6-32
NSSTART verb 6-33
NSSTOP verb 6-34

O

one-way bracket
  See pipeline transaction

P

partner LU definition 6-6
performance strategies
  improving performance of CPI-C applications 3-1, 3-17
permission to send 3-16, 3-17
pipeline transaction 3-6
Program Launcher 5-2, 5-7, 5-12
pseudonym files
  in C 2-6

Q

queue-level non-blocking 4-1
queued Attach 5-2, 5-7

R

RECEIVE (CMRCV)
  CM_RECEIVE_AND_WAIT receive_type 4-2
  CM_RECEIVE_IMMEDIATE receive_type 4-2
Receive call 2-7, 3-19
receiving data 3-15
rejecting an Attach 5-8
remote procedure call
  database application 3-7
  status queries 3-7
removing an Attach from the Attach list 5-7
Request and Reply transaction
  See inquiry transaction
Request_To_Send (CMRTS) 3-18
return code 5-8

S

SAA/LU 6.2: Distributed Networks and Applications X-1
sample programs 2-6
Send_Data (CMSEND) 2-7, 3-15, 3-18, 3-19, 4-3
Send_Error (CMSERR) 3-9, 3-16, 3-17, 3-18, 4-2, 4-4
server program, cascading 3-15
session
  activating 6-6
Set_Log_Data (CMSLD) 2-7, 2-9
Set_Mode_Name (CMSMN) 2-8
Set_Send_Type (CMSST) 3-18
Set_TP_Name (CMSTPN) 5-11
short conversation transaction
  See pipeline transaction
side information
  defining 2-8
SNASVCMG mode name 2-8
stack size 2-7
starting a program 3-12
starting programs 5-12
Systems Network Architecture Technical Overview X-1

T

terms used in this book xii
The APPC Resource Book X-1
TP definitions 5-3
TP name
  See transaction program (TP) name
TP_name characteristic 2-8
TPF
  See Transaction Processing Facility (TPF)
transaction processing application model 3-3
Transaction Processing Facility (TPF) 3-3
transaction program (TP)
  designing 3-1
transaction program (TP) name
  local 5-11
transaction types
  credit-check 3-8
  database-update 3-11
  file-transfer 3-10
  inquiry 3-7
  pipeline 3-6
  realtime 3-4

U

UNBIND X-13
user session 6-6
  defined X-13

V

verb, node operator API
  NSACT 6-6
  NSATTACH 5-6, 6-10
  NSCONLAN 6-11
  NSCONMPT 6-13
  NSCONREM 6-14
  NSCONTWX 6-16
  NSDEACT 6-17
  NSDEASES 6-18
  NSDEFMO 6-19
  NSDEFSI 6-21
  NSDEFTP 6-24
  NSDELMO 6-20
  NSDELSI 6-23
  NSDELTP 6-26
  NSDISCON 6-27
  NSINFO 6-28
  NSREJATT 6-32
  NSSTART 6-33
  NSSTOP 6-34

W

Wait_For_Completion (CMWCMP) 5-11
waiting indefinitely
  understanding 4-2
  waiting after a deallocate or Send_Error call 4-4
  waiting for confirmation 4-4
  waiting for data 4-2
  waiting for incoming Attach 4-3, 5-9, 5-10
  waiting for pacing 4-3
Windows, Microsoft
  using with Networking Services 2-11

Communicating Your Comments to IBM

APPC Networking Services for Windows
Application Programmer's Reference
Version 1.0

Publication No. SC31-8126-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

If you prefer to send comments by mail, use the RCF at the back of this book.

If you prefer to send comments by FAX, use this number:

United States and Canada: **1-800-227-5088**

If you prefer to send comments electronically, use this network ID:

- IBM Mail Exchange: **USIB2HPD at IBMMAIL**
- IBMLink: **CIBMORCF at RALVM13**
- Internet: **USIB2HPD@VNET.IBM.COM**

Make sure to include the following in your note:

Title and publication number of this book
Page number or topic to which your comment applies.

Help us help you!

APPC Networking Services for Windows
Application Programmer's Reference
Version 1.0

Publication No. SC31-8126-00

We hope you find this publication useful, readable and technically accurate, but only you can tell us! Your comments and suggestions will help us improve our technical publications. Please take a few minutes to let us know what you think by completing this form.

Overall, how satisfied are you with the information in this book?	Satisfied	Dissatisfied

How satisfied are you that the information in this book is:	Satisfied	Dissatisfied
Accurate		
Complete		
Easy to find		
Easy to understand		
Well organized		
Applicable to your task		

Specific Comments or Problems:

Please tell us how we can improve this book:

Thank you for your response. When you send information to IBM, you grant IBM the right to use or distribute the information without incurring any obligation to you. You of course retain the right to use the information in any way you choose.

Name

Address

Company or Organization

Phone No.

Help us help you!
SC31-8126-00

Cut or Fold
Along Line

IBM®

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department F94/Building 673
PO BOX 12195
RESEARCH TRIANGLE PARK, NORTH CAROLINA 27709-2195

Fold and Tape **Please do not staple** Fold and Tape

SC31-8126-00

Cut or Fold
Along Line

IBM®

Program Number: 5871-AAA 9737

Printed in U.S.A.

SC31-8126-