ACSE/Presentation and ROSE Interface Programmer's Guide

Edition 4

HEWLETT® PACKARD

32070-90032
HP9000 Networking
05/97

Legal Notices

Contents

Contents

Contents

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

Edition 1	March 1992
Edition 2	January 1995
Edition 3	July 1996
Edition 4	May 1997

In This Book

This manual describes tools and procedures for using the ACSE/Presentation and ROSE application programmatic interface for OSI products. It is divided into three chapters, which provide the following information:

Chapter 1	“APRI Overview” introduces the ACSE/Presentation and ROSE Interface and the supported standards.
Chapter 2	“Supported ACSE/Presentation and ROSE Calls” contains descriptions of the programmatic calls supported by ACSE/Presentation and ROSE.
Chapter 3	“Using ACSE/Presentation” provides a sample session using ACSE/Presentation between two processes in synchronous mode.
Chapter 4	“Programming Guide” provides information to assist you in writing and executing applications using the ACSE/ Presentation (A/P) interface.
Chapter 5	“Troubleshooting Your Application” describes API tracing for ACSE/Presentation followed by API tracing for ROSE.
Appendix A	“ACSE/Presentation Reference Pages” includes a list of the supported ACSE/Presentation attributes and a glossary of terms.

1 APRI Overview

This chapter provides an overview of what the HP ACSE/Presentation and ROSE interface provides.

Systems Supported

Access to the ACSE/Presentation and ROSE programmatic interface is provided with the HP OTS/9000 product on HP 9000 systems.

ASN.1 Support

If your application requires encoding/decoding abstract syntax notation one (ASN.1) data structures, you may need to use an ASN.1 compiler. Refer to your compiler's documentation for information about using ASN.1 in your application programs.

File Naming

To use the MAN feature on systems that support a maximum file length of 14 characters, the ACSE/Presentation and ROSE calls with longer names have short alias names you can use. The short names are listed in the tables in chapter 2.

What is the HP ACSE/Presentation and ROSE Interface

The ACSE/Presentation and ROSE interface (APRI) provides a programmatic interface to the Association Control Service Element (ACSE), Remote Operation Service Element (ROSE) and Presentation layer protocols over an OSI network. See Figure 1-1.

ACSE/Presentation

Using the ACSE/Presentation (A/P) interface enables two or more application processes on the same or different computers to:

- Establish an association (connection) with another application process
- Exchange (send and receive) information and
- Shutdown the association (connection)

ROSE

Using ROSE with ACSE/Presentation provides the request/reply service which is useful in building distributed applications. Note that ROSE cannot be used independently of the ACSE/Presentation interface.

Support for Multi-Threaded Applications

This version of HP OTS/9000 supports multi-threaded applications to be written using the same programmatic interface as before for APLI and ROSE. Applications can use either DCE User Threads or Kernel threads interfaces. The following programming guidelines need to be followed to be able to write multi-threaded applications:

- The application should be compiled with the `-D_REENTRANT` compiler flag. Also, it may use compiler flag `-D_PTHREADS_DRAFT4` for linking with DCE User Threads library or `-D_KERNEL_THREADS` for Kernel threads library.
- Multi-threaded applications must define ap_errno as:

  ```
  extern unsigned long _ap_errno();
  ```

```
#define ap_errno _ap_errno()
```

- Any multi-threaded application program, which also includes osi_lib.h, should include it after the threads-specific include files.
- See the appropriate man page regarding thread-safe information about the various api's.

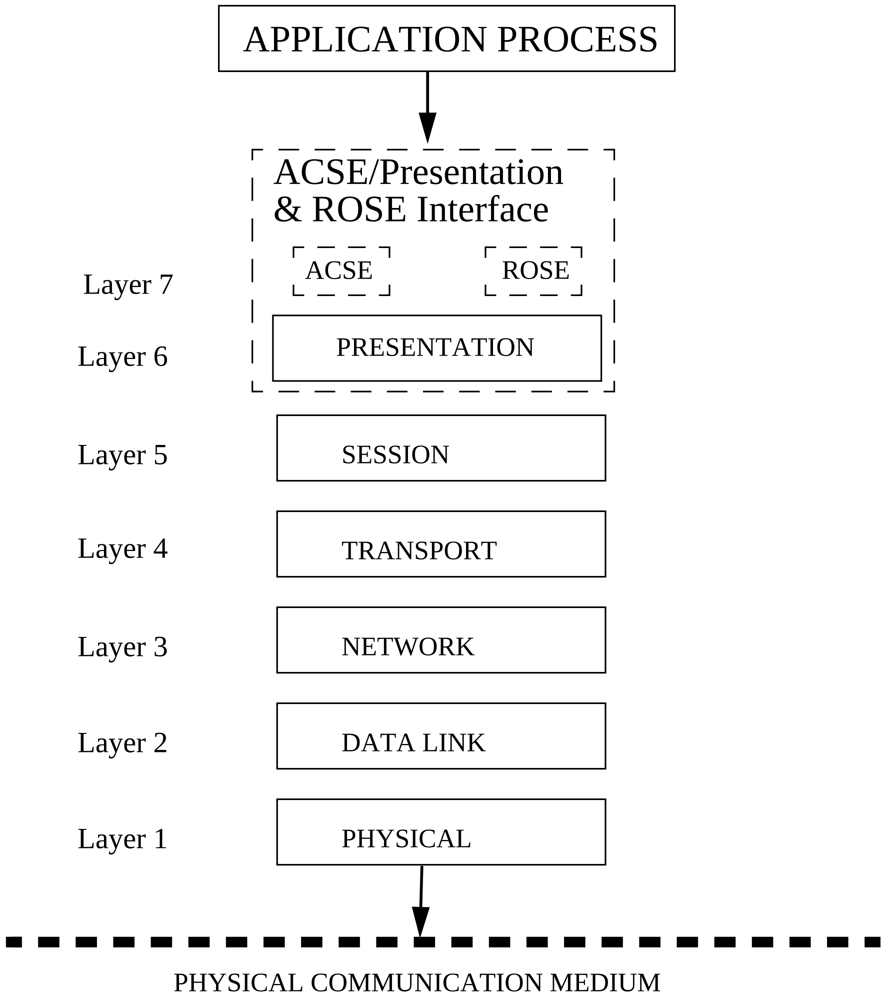
Standards Supported

The interface provided is based on the services defined in the ISO ACSE and Presentation Service Definitions (ISO 8649 and ISO 8822) and ROSE (ISO 9072).

The HP OTS/9000 subset implementation of the ACSE/Presentation library interface is based on the UNIX International OSI ACSE/Presentation Library Interface specification draft dated October 25, 1990.

HP OTS/9000 also supports a restricted mode of operation which allows internetworking with a system that conforms to the ITU-T Recommendation X.410 (1984).

Figure 1-1 **OSI MODEL - ACSE Presentation and ROSE**



Limitations

A subset of the standard has been implemented with the following limitations.

For limitations on particular parameters, please refer to the manpages for the A/P and ROSE calls.

ACSE/Presentation:

- The ap_restore(), ap_save() and ap_osic() calls are not supported.
- The environment file (env_file) is not supported. The user's application environment is initialized using ap_init_env(), after which individual attributes can be set or changed using ap_set_env().
- Setting multiple environment variables with one call to ap_set_env() is not supported.
- Retrieving (getting) multiple environment variables with one call to ap_get_env() is not supported.
- The ap_env() attributes AP_DPCN, AP_DPCR, AP_QLEN and AP_QOS are not supported. The AP_STREAMS_FLAGS options AP_HUP_ONABORT and AP_PEEK are not supported.
- For ap_rcv(): The AP_ALLOC option is not supported. The buffer pointed to by the ubuf argument must be allocated by the user. The AP_VECT option is not supported. In struct osi_buf, the next field is ignored. The ubuf argument points to a single buffer.
- For ap_snd(): The AP_VECT option is not supported. In struct osi_buf, the next field is ignored. The ubuf argument points to a single buffer. The AP_DELAY option is not supported. Extended concatenation is not supported.

  The A_ASSOC_REQ can fail with AP_AGAIN. A_ASSOC_REQ is not subject to flow control.
- ACSE/presentation primitives used in ACSE/Presentation calls: A subset of the defined primitives are supported. Chapter 2 provides a list of the supported primitives.

ROSE:

- Reliable Transfer Service (RTS) including operation class is not supported.
- Priority is not implemented for this release.

Who Should Use This Manual

This manual is provided for application programmers who need to use ACSE/Presentation and ROSE services.

It is expected that the user of this manual is an experienced applications programmer with knowledge of the HP-UX operating system and programming environment, the X.25 protocol, OSI transport, session, and presentation layer functions, networking concepts and the following list of documents:

- ISO 7498 - Information Processing Systems - Open Systems Interconnection - Reference Model of Open System Interconnection
- ISO 8649 - Information Processing Systems - Open Systems Interconnection - Service Definition for the Association Control Service Element
- ISO 8650 - Information Processing Systems - Open Systems Interconnection - Protocol Specification for the Association Control Service Element
- ISO 8822 - Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Service Definition
- ISO 8823 - Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Protocol Specification
- ISO 8824 - Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax One (ASN.1)
- ISO 8825 - Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax One (ASN.1)
- NIST - Stable Implementation Agreements for Open Systems Interconnection Protocols
- Overview of the Application Service Library Model
- ISO 9072/1 Remote Operations Service Definition
- ISO 9072/2 Remote Operations Protocol Specification

2 Supported ACSE/Presentation and ROSE Calls

This chapter provides lists and brief descriptions of the supported ACSE/ Presentation and ROSE calls and primitives.

ACSE/Presentation Calls and Primitives

The primitives are grouped in the sequence used in an application. For example, the A_ASSOC_xxx primitives are listed in the order:

```
A_ASSOC_REQ (request)
A_ASSOC_IND (indication)
A_ASSOC_RSP (response)
A_ASSOC_CNF (confirmation)
```

In the manpages entitled ap_intro and ap_env, you'll find introductions to the A/P environment and the A/P library environment attributes. They also supply detailed descriptions or references made in this manual. The manpages are accessible online with the manpage function of HP-UX.

Included here are tables of attributes that provide:

- Name and purpose of each attribute
- Data types legal for the attribute
- Default values supplied with attribute (if any)
- Values legal for the attribute (if applicable)
- Readable states for the attribute (as values of the AP_STATE attribute)
- Writable states for the attribute (as values of the AP_STATE attribute)

The following lists A/P functions first by task, then by call in alphabetic order in Table 2-1. The supported primitives are described in Table 2-2.

A/P Functions

The A/P calls can be separated into functions that perform the following tasks:

- Establish/release the communication endpoint:

  ```
  ap_open()
  ap_close()
  ```

- Manage the ACSE/Presentation environment:

```
ap_init_env()
ap_set_env()
ap_get_env()
```

- Send and receive service primitives which includes sending/receiving service requests and sending/receiving user data:

```
ap_snd()
ap_rcv()
```

- Manage your applications:

```
ap_error()
ap_free()
ap_poll()
```

ACSE/Presentation Calls

The following table lists the supported ACSE/Presentation function calls.

Table 2-1 ACSE/Presentation Calls

A/P Call	Description
`ap_close()`	Release a communication endpoint.
`ap_error()`	Returns an error message.
`ap_free()`	Free memory for the A/P data structures.
`ap_get_env()`	Retrieve the value of an A/P environment attribute.
`ap_init_env()`	Initialize the A/P environment.
`ap_open()`	Establish a communication endpoint to support an instance of the A/P environment.
`ap_poll()`	Provides an interface for detecting events on communication endpoints.

A/P Call	Description
`ap_rcv()`	Receive an A/P primitive over an association. User data may be associated with the primitive.
`ap_set_env()`	Set an attribute in the A/P environment.
`ap_snd()`	Send an A/P primitive over an association. User data may be associated with the primitive.

ACSE/Presentation Primitives

The following table lists the supported ACSE/Presentation layer service primitives. These primitives are used with the ap_rcv() and ap_snd() calls as described below.

Some of the primitives contain a shorter name in BOLD under the primitive's name. On systems that support a maximum length of 14 characters for file names, use the name listed in bold with the MAN feature. Primitives with no short name can be used on all systems.

The primitives are grouped in the sequence used in an application. For example, the A_ASSOC_xxx primitives are listed in the order:

```
A_ASSOC_REQ (request)
A_ASSOC_IND (indication)
A_ASSOC_RSP (response)
A_ASSOC_CNF (confirmation)
```

Table 2-2 ASCE/Presentation Primitives

Primitives	Description
`A_ABORT_REQ`	Used with ap_snd() to request the abnormal release of an association.
`A_ABORT_IND`	Used with ap_rcv() to indicate the abnormal release of an association.
`A_ASSOC_REQ`	Used with ap_snd() to initiate establishing an association.

Primitives	Description
A_ASSOC_IND	Used with ap_rcv() to indicate a request for association establishment.
A_ASSOC_RSP	Used with ap_snd() to respond to an association establishment request.
A_ASSOC_CNF	Used with ap_rcv() to confirm the establishment of an association.
A_PABORT_REQ	Used with ap_snd() to initiate a presentation layer provider abort. Provides the option of aborting when an invalid PDU is received.
A_PABORT_IND	Used with ap_rcv() to indicate an association has been abnormally released because of problems below the application layer.
A_RELEASE_REQ A_RELEASEREQ	Used with ap_snd() to request the normal release of an association.
A_RELEASE_IND A_RELEASEIND	Used with ap_rcv() to indicate that the remote service user wants to release the association.
A_RELEASE_RSP A_RELEASERSP	Used with ap_snd() to respond to an association release request.
A_RELEASE_CNF A_RELEASECNF	Used with ap_rcv() to confirm the acceptance or rejection of a previously sent release request.
P_DATA_REQ	Used with ap_snd() to send normal user data.
P_DATA_IND	Used with ap_rcv() to indicate the receipt of user data.
P_RESYNC_REQ	Used with ap_snd() to issue a resynchronized request.
P_RESYNC_IND	Used with ap_rcv() to indicate a resynchronized request.
P_RESYNC_RSP	Used with ap_snd() to respond to a resynchronized request.

Primitives	**Description**
P_RESYNC_CNF	Used with ap_rcv() to confirm a resynchronized request.
P_SYNCMINOR_REQ P_SMINOR_REQ	Used with ap_snd to request the setting of a minor sync point.
P_SYNCMINOR_IND P_SMINOR_IND	Used with ap_rcv to indicate a request the to set a minor sync point.
P_SYNCMINOR_RSP P_SMINOR_RSP	Used with ap_snd to respond to a minor sync point
P_SYNCMINOR_CNF P_SMINOR_CNF	Used with ap_rcv to confirm a sync minor request.
P_TOKENGIVE_REQ P_TOKENGIREQ	Used with ap_snd() to give tokens to another session user.
P_TOKENGIVE_IND P_TOKENGIIND	Used with ap_rcv() to indicate the receipt of newly acquired session tokens.
P_TOKENPLEASE_REQ P_TOKENPLREQ	Used with ap_snd() to request session tokens.
P_TOKENPLEASE_IND P_TOKENPLIND	Used with ap_rcv() to indicate a request for tokens.

ROSE Calls and Primitives

The supported ROSE function calls and primitives are listed in Table 2-3 and in Table 2-4. Also refer to the man page RO_INTRO(5) for general information about ROSE.

ROSE Calls

ROSE only supports two calls as listed below. ROSE is used to provide ROSE provider services to an application using the ACSE/Presentation interface.

Table 2-3 Rose Calls

ROSE Calls	Description
`ro_bind()`	Enable ROSE provider.
`ro_unbind()`	Disable ROSE provider.

ROSE Primitives

ROSE primitives are listed in the following table. Note that the primitives are grouped in the sequence used in an application. For example, the RO_RESULT_xxx primitives are listed RO_RESULT_REQ (request) followed by RO_RESULT_IND (indication).

Table 2-4 ROSE Primitives

Primitives	Description
`RO_ERROR_REQ`	Negative remote operation results request.
`RO_ERROR_IND`	Negative remote operation result indication.
`RO_INVOKE_REQ` `RO_INVOKEREQ`	Remote operation request.
`RO_INVOKE_IND` `RO_INVOKEIND`	Remote operation indication.

Primitives	Description
`RO_REJECTP_IND` `RO_REJ_P_IND`	ROSE provider rejection indication.
`RO_REJECTU_REQ` `RO_REJ_U_REQ`	ROSE user invocation rejection request.
`RO_REJECTU_IND` `RO_REJ_U_IND`	ROSE user invocation rejection indication.
`RO_RESULT_REQ` `RO_RESULTREQ`	Positive remote operation results request.
`RO_RESULT_IND` `RO_RESULTIND`	Positive remote operation results indication.

3 Using ACSE/Presentation

This chapter provides a sample session using ACSE/Presentation between two processes in synchronous mode.

Synchronous ACSE/Presentation

ACSE/Presentation calls block until the call is complete in synchronous mode. See the section, "Using Synchronous vs. Asynchronous Mode" in chapter 4 for more information. In order to use the ACSE/Presentation service, the processes do the following:

1. Create a communication endpoint.
2. Establish an association that requires:
   a. Initializing the ACSE/Presentation environment.
   b. Establishing the roles of each process for the connection, one as initiator, the other as responder.
   c. Binding the communication endpoint to a presentation address (p-selector, s- selector, t-selector, and NSAP).
   d. Setting up the data transfer environment.
   e. Requesting an association
   f. Confirming the association.
3. Exchange data in the agreed upon context and transfer syntax
4. Release the association:
   a. Request an association release.
   b. Confirm a release request.
5. Close down the communication endpoint.

The following pages describe these steps in more detail.

Summary of Calls

A summary of the ACSE/Presentation calls are described in the following sections.

Table 3-1 **ACSE/Presentation Call Summary**

Process A	**Process B**
ap_open()	ap_open()
ap_init_env()	ap_init_env()
ap_set_env() (*initiator*)	ap_set_env() (*responder*)
ap_set_env() (*p-address*)	ap_set_env() (*p-address*)
ap_set_env() (*B's p-address*)	
ap_set_env() (*appl. context*)	
ap_set_env() (*PCDL*)	
ap_snd() (*assoc. req.*) =====>	ap_poll()
	ap_rcv() (*assoc. ind.*)
	ap_get_env() (*appl. cnxt.*)
	ap_set_env() (*PCDRL*)
ap_poll() <======	ap_snd() (*assoc. rsp.*)
ap_rcv() (*assoc. cnf.*)	
ap_snd() (*data*) ======>	ap_poll()
	ap_rcv() (* data *)
ap_snd() (*rel. req.*) ======>	a p _poll()

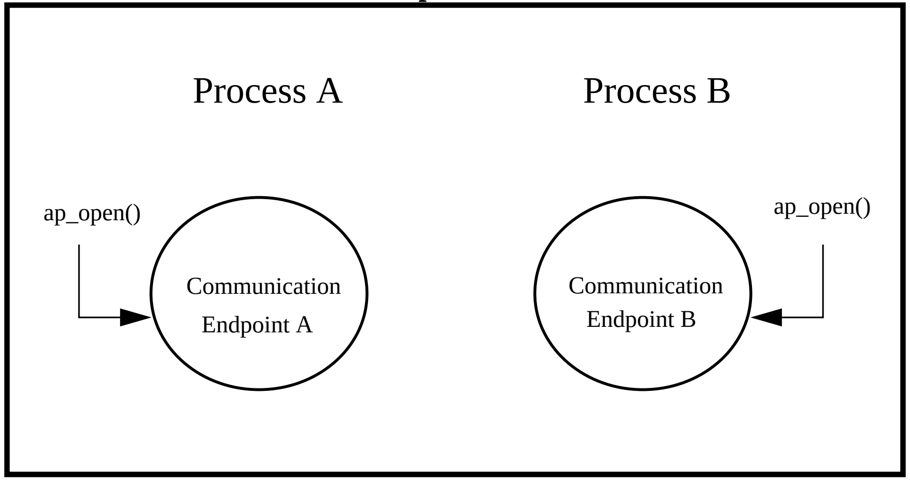
Process A	**Process B**
	`ap_rcv() (*`*rel. ind.*`*)`
`ap_poll() <======`	`ap_snd() (*`*rel. rsp*`*)`
`ap_rcv() (*`*rel. cnf.*`*)`	
`ap_close()`	`ap_close()`

Step 1: Create a Communication Endpoint

Process A and Process B must both create communication endpoints as shown in Figure 3-1, using the call ap_open(). A communication endpoint is a file descriptor (fd) which is returned on successful completion of the ap_open() call. The process uses the file descriptor in all subsequent A/P calls.

Each process must create a communication endpoint, and only one association can be associated with each endpoint. To create multiple associations, you must create additional communication endpoints for each one.

Figure 3-1 Create a Communication Endpoint



Step 2: Establish an Association

The steps to establish an association are described below.

Initialize the A/P Environment

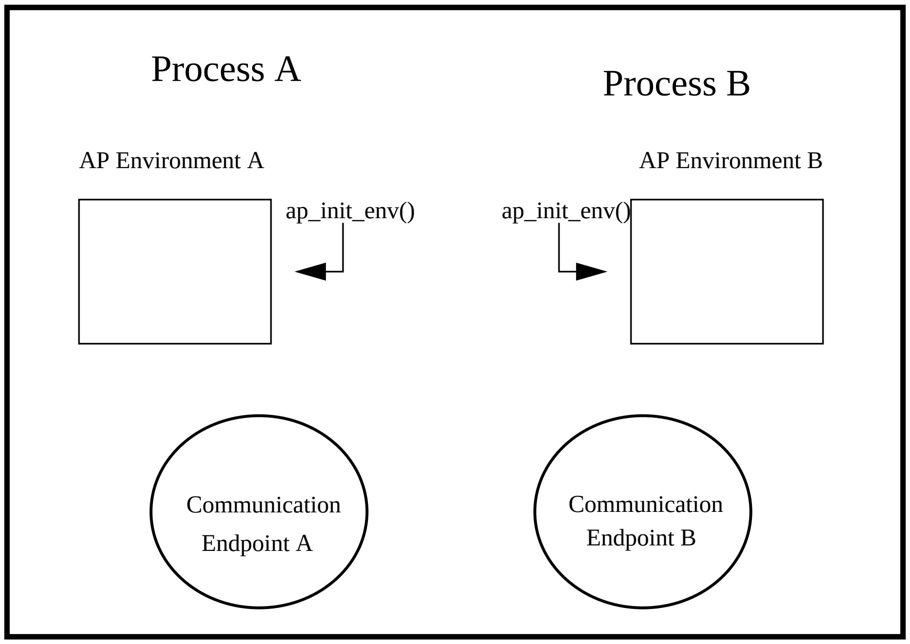
To use the A/P services, each process must initialize the ACSE/Presentation environment by calling ap_init_env(). (See Figure 3-2.)

The ap_init_env() call allocates memory for the environment attributes, and sets the attributes to default values. (See the AP_ENV Table of Attributes in Appendix A for more information.)

Since this implementation does not support the environment file, the env_file parameter is ignored.

To modify defaulted attribute values, the ap_set_env() call is used as shown in the following steps.

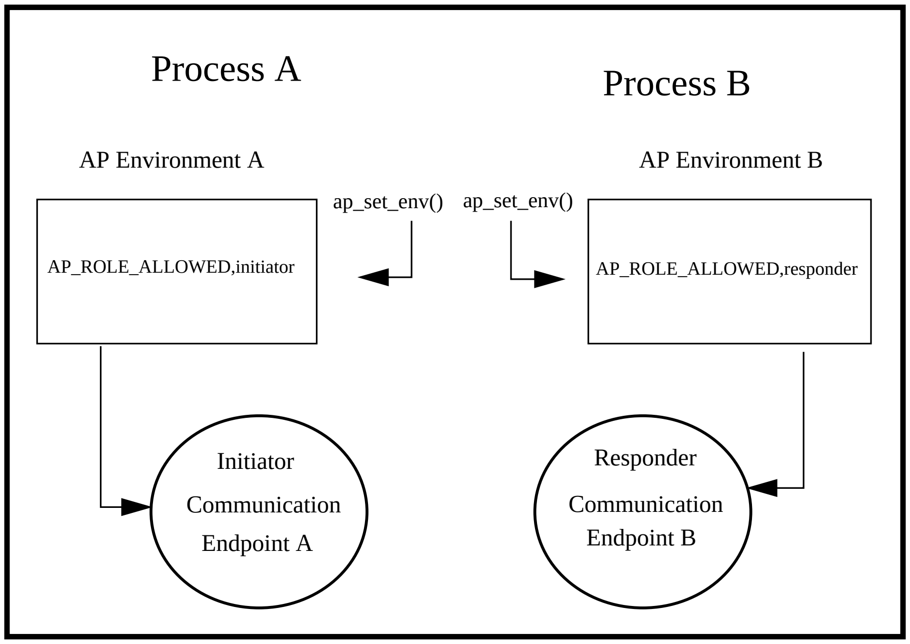
Figure 3-2 **Initialize Environment**



Establish Initiator and Responder Roles

Process A and Process B establish initiator and responder roles using the attribute, AP_ROLE_ALLOWED in a call to ap_set_env(). In this example, Process A is the initiator, and Process B is the responder. (See Figure 3-3.)

Figure 3-3 Establish Roles



NOTE The initiator and responder roles must be established before binding the communication endpoint to the local presentation address while in the AP_UNBOUND state. Any change in AP_ROLE_ALLOWED after binding is ignored.

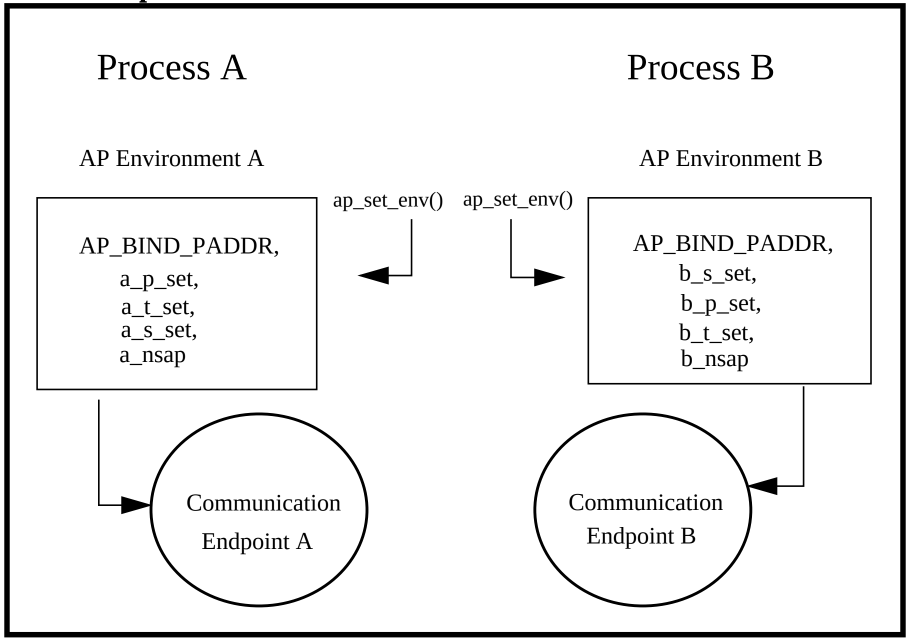
Binding the Endpoints

Process A and Process B must each bind a valid presentation address (p-selector, s-selector, t-selector, and NSAP) to its endpoint by using the attribute AP_BIND_PADDR in the ap_set_env() call. (See Figure 3-4.) For more information on presentation addresses, refer to Chapter 4.

APRI also supports the binding of the local presentation address without specifying a network address. This allows a single responder to receive association indications for any of the subnetworks configured on an OTS/9000 system. To use this feature, when setting the local address through the AP_BIND_PADDR environment attribute, set the n_nsaps field of the ap_paddr_t structure to zero. When this feature is used, no local NSAP information need be provided in the ap_paddr_t structure.

NOTE Rebinding the presentation address to a communication endpoint is not supported in this release. Binding to the presentation address can only be done in the AP_UNBOUND state.

Figure 3-4 **Bind Endpoint to Presentation Address**

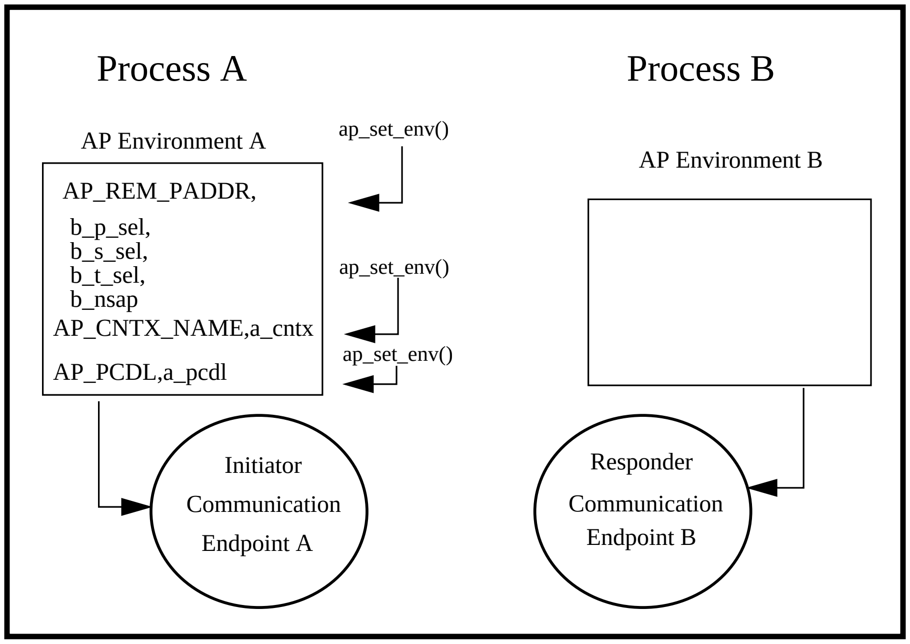


Set up the Data Transfer Environment

In order for Process A to communicate with Process B, Process A calls ap_set_env() using the AP_REM_PADDR attribute which contains the presentation address of Process B. (See Figure 3-5.)

Process A calls ap_set_env() using the AP_CNTX_NAME attribute to identify the application context name. Process A then calls ap_set_env() using the AP_PCDL attribute to propose transfer syntaxes for each proposed abstract syntax that will be used for data transfer between Process A and B.

Figure 3-5 **Establish Data Transfer Environment**

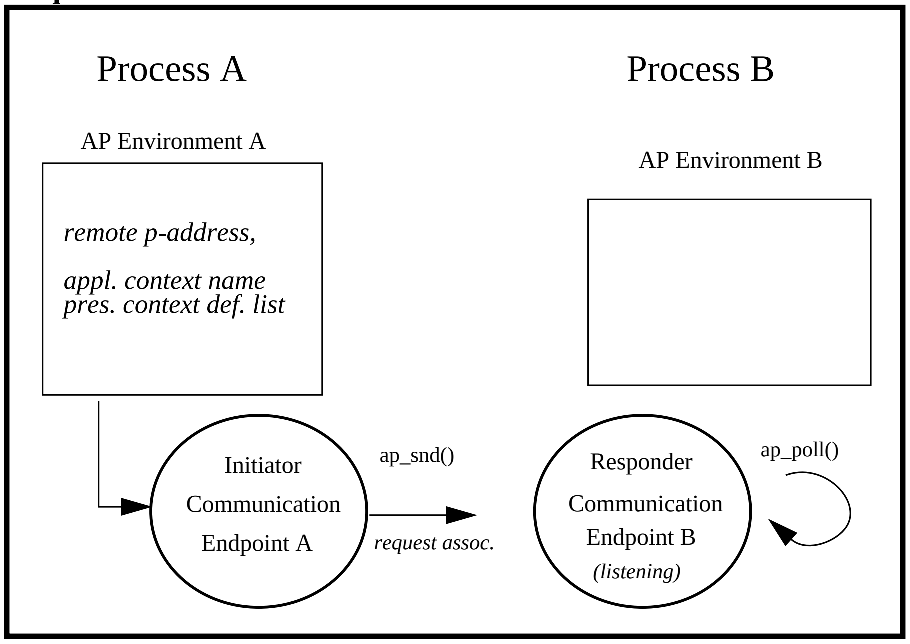


Request the Association

Process ap_snd() uses the A_ASSOC_REQ primitive and includes the application context name, presentation context definition list (PCDL), and Process B's presentation address. Process B uses ap_poll() to listen for the association request. (See Figure 3-6.)

This can be used to limit the time Process B can wait for an association request.

Figure 3-6 **Request Association**



Confirm the Association

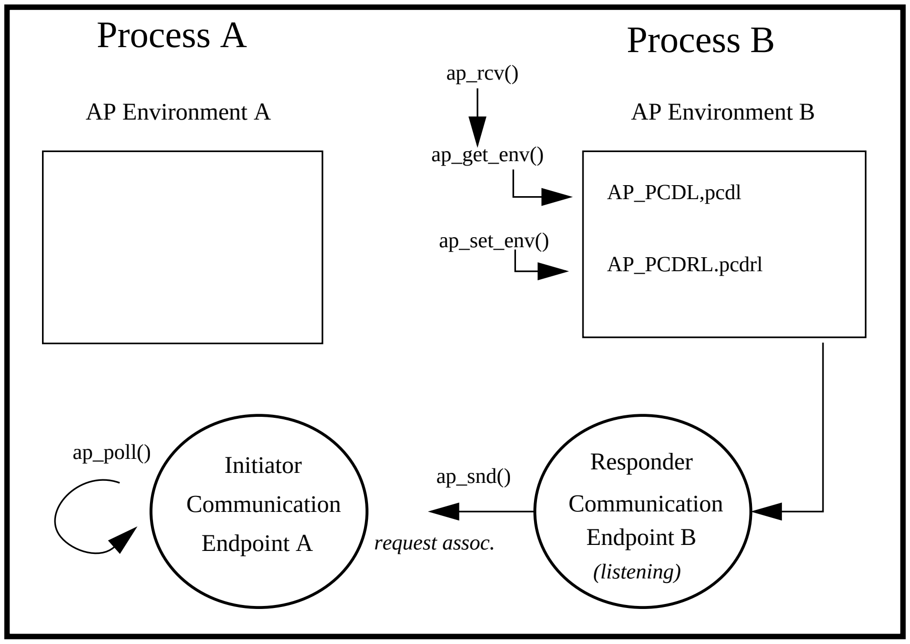
When a request has been received by checking the ap_poll() events parameter, Process B calls ap_rcv() and receives A_ASSOC_IND. Process B must find out what the proposed transfer syntaxes are by calling ap_get_env() using the AP_PCDL attribute.

It checks the transfer syntax to Process A's proposed syntax. If Process B determines that the syntax is valid, Process B calls ap_set_env() using the AP_PCDRL attribute to set its transfer syntax. Process B sets the res field in the structure ap_cdrl_t to ACCPT and the transfer syntax pointer to the supported transfer syntax object id. Process B then calls ap_snd() using the A_ASSOC_RSP primitive to accept the association request. (See Figure 3-7.)

Process A must wait for Process B to respond with the connection response by using ap_poll() or by making a synchronous ap_rcv() call.

Process B can reject the connection if the proposed transfer syntax is not supported by setting the res field in structure ap_cdrl_t to USER_REJ or PROV_REJ and prov_rsn is the reason for the reject. If the process cannot accept the association request for other reasons, the process sets the reason using the A_ASSOC_RSP primitive.

Figure 3-7 **Confirm Association**



Step 3. Exchanging Data

Once an association is established, processes can send and receive control data (primitives) and user data.

It is the responsibility of the process to encode and decode data with the selected transfer syntax. For complex data types, you can choose to use an ASN.1 compiler to assist in the creation of these routines.

Process A retrieves its supported and negotiated transfer syntaxes by calling ap_get_env() using the AP_DCS attribute, and to find out the result of each proposed presentation context by calling ap_get_env() using AP_PCDRL to look at the res and prov_rsn fields. (See Figure 3-8.)

Process A is now ready to send data using the P_DATA_REQ primitive in the ap_snd() call in the agreed upon presentation context.

Process B uses ap_poll() to monitor events. The data received by ap_rcv() is indicated by the P_DATA_IND primitive.

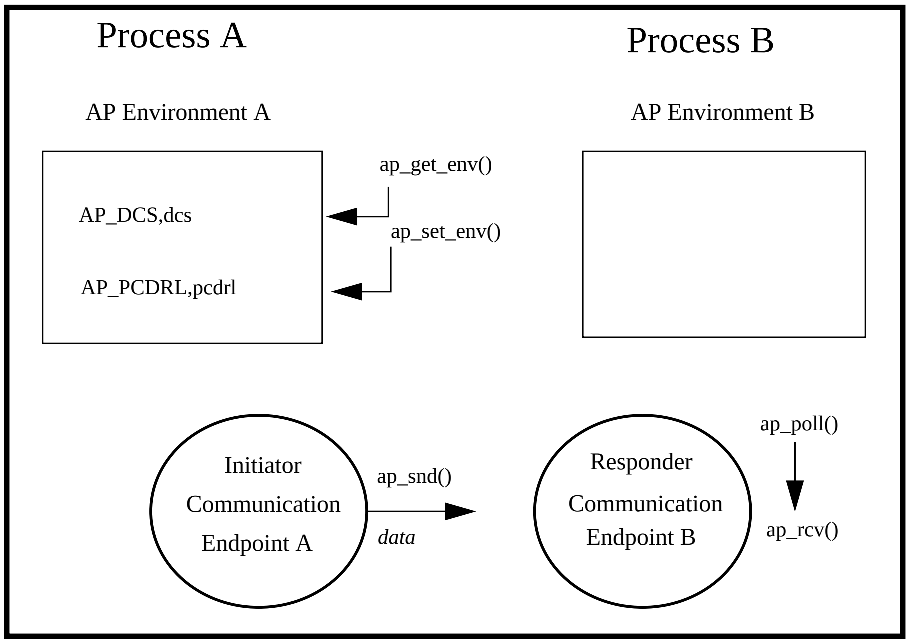
Note that the AP_ALLOC option is not supported. The buffer pointed to by ubuf must be allocated by the user.

If the data received is more than the allocated buffer, the flags parameter AP_MORE bit is set. If set, continue to invoke ap_rcv() until the AP_MORE bit is off.

Check the sptype parameter each time ap_rcv() is called. Some primitives such as A_ABORT_IND are not flow-controlled and may interrupt the receipt of a partially received primitive (for example, AP_MORE bit is set). If this occurs, any remaining data from the previous primitive is lost.

NOTE Note that handling expedited data is not supported in this release.

Figure 3-8 **Send Data**



Step 4: Release the Association

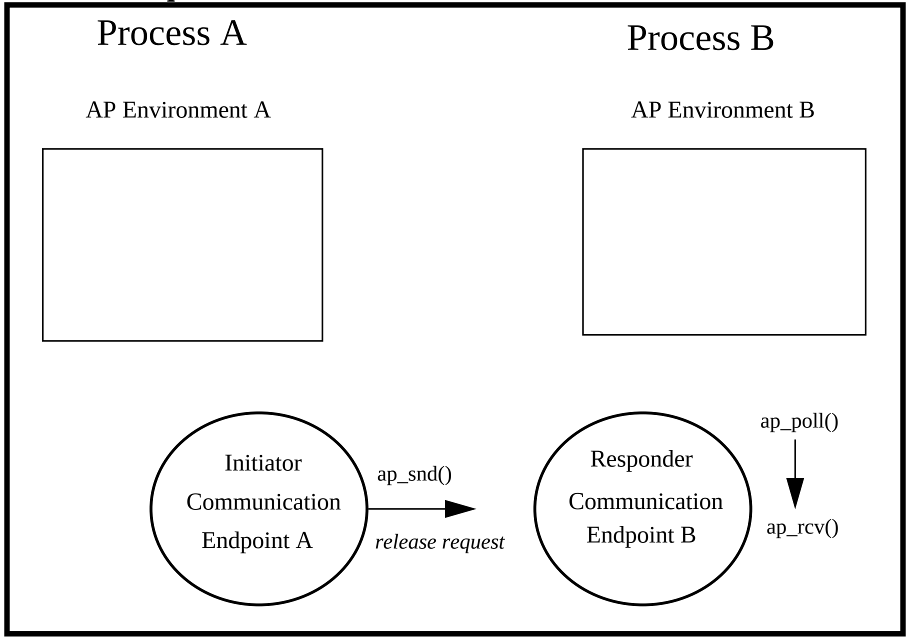
Processes must cooperate to ensure that no data is lost when the association is released. This can be accomplished by requesting a release, accepting the release, then terminating the association.

Requesting the Release

After Process A has completed sending user data, it sends a release request to Process B using ap_snd() with the A_RELEASE_REQ primitive (Figure 9).

Process B uses ap_poll() to monitor events, and calls ap_rcv() to receive the release request.

Figure 3-9 **Release Request**



Confirm the Release Request

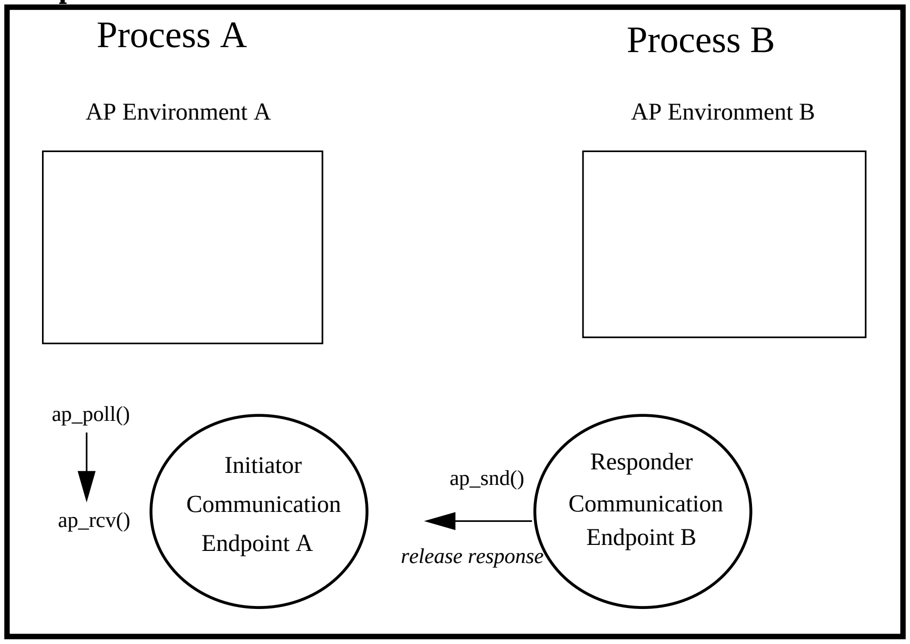
Process B accepts the request and sends a positive response using the ap_snd() call with the A_RELEASE_RSP primitive (Figure 3-10).

The process closes the association until it receives the positive release response from Process B.

Process A monitors events with ap_poll() and receives the response by calling ap_rcv().

If an association is released abnormally, for example, if a lower layer problem occurs, a provider abort (A_PABORT_IND primitive) may be received.

Figure 3-10 Response to Release



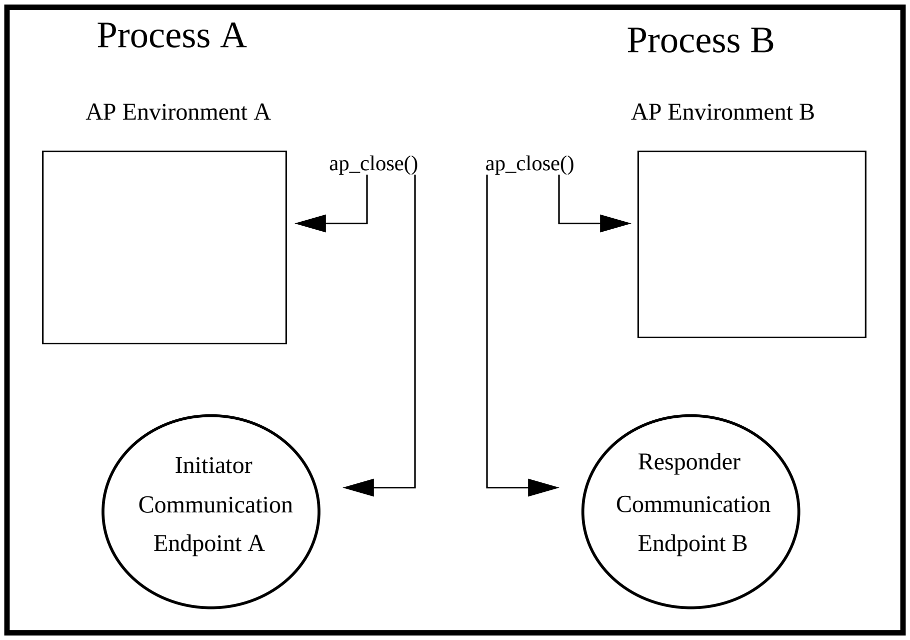
Step 5: Closing the Communication Endpoint

Finally, both processes use ap_close() to close down the communication endpoint (file descriptor) and A/P environment resources (see Figure 3-11). If the association is still alive when ap_close() is invoked, the association will be aborted by the provider before the communication endpoint is released.

When an association is terminated, you can re-use the communication endpoint for a new association. ap_close() is used when there is no further use for the communication endpoint.

CAUTION Resources allocated upon the initial ap_open() will not be released unless an explicit ap_close() call is made for each endpoint or the process is terminated.

Figure 3-11 **Close the Association**



4 Programming Guide

This chapter provides information to assist you in writing and executing applications using the ACSE/Presentation (A/P) interface.

Summary of Programming Tasks

The following list summarizes the tasks you need to perform in order to successfully create and execute your programs. These tasks are discussed in more detail in the following sections.

1. Prepare system. Make sure HP OTS/9000 h as been installed, configured, and started successfully. (See the *Installing and Administering OSI Transport Services* manual for information.)
2. Manage the A/P environment. To establish and maintain connections, attributes must be set correctly:
   a. Determine which process will be the initiator and which will be the responder. For example, one process will initiate the association, the other will accept the association. By default, either process can be initiator or responder.
   b. Determine how addressing will be handled. Valid presentation addresses must be configured for ACSE/Presentation applications.
   c. Determine the presentation context and application context that will be used. Processes must agree on both the context and transfer syntax they will use.
   d. Determine portability and future expansion requirements. Review considerations that will assist in migrating as the standards for ACSE/ Presentation evolve and are accepted.
3. Manage data. Understand how data handling is done by the A/P interface.
   - How control data is used for conveying additional information to the receiving process as part of the A/P primitive.
   - Handle data in multiple sends and receives.
   - Determine user data encryption requirements for the application. Must be handled by the application. You may need to use an ASN.1 compiler to assist you.
4. Decide if you will use synchronous and/or asynchronous mode processing.

5. Decide if multiple associations need to be supported. The process accepting multiple connections needs to create additional communication endpoints and instances of the A/P environment. Review the paragraphs describing: execution mode, connection retry, and maximum number of connections supported (system, processes, and other applications).
6. Other tasks:
   - Understand how file descriptors, common parameters, data, and structures are used in ACSE/Presentation applications.
   - Handle signals. You need to create signal handler routines to capture unexpected program interrupts.
   - Handle errors. ACSE/Presentation calls return values indicating success or failure, and also return specific error conditions that the processes must handle.
7. Prepare to start your applications. Before running:
   - Determine how the remote processes will be started.
   - Check the system before executing applications.
   - Start OTS before executing ACSE/Presentation applications.
   - If you install a new version of HP OTS/9000, you must re-link your programs with the new library.
8. Troubleshoot your application. If you have problems, you can use API tracing described in chapter 5, “Troubleshooting your Application.”

1. Prepare the System

Before you use the ACSE/Presentation interfaces, make sure that the following has been done:

- Verify HP OTS/9000 h as been installed correctly.
- Check that the presentation addresses, in particular, the local NSAP(s) used by the ACSE/Presentation applications that have been configured in the OTS configuration file.
- Verify that HP OTS/9000 h as been started either manually using osiadmin or at start-up time.

2. Manage the A/P Environment

The ACSE/Presentation environment contains the necessary information to establish and maintain an association. The primitives used to identify this information are called attributes. See Appendix A, “AP_ENV Table of Attributes” which describes the states and whether or not particular attributes can be written to or read from.

Three A/P calls are used to manage information in the A/P environment:

- ap_init_env() establishes the environment and sets up default values for many of the attributes. Note that the env_file variable in ap_init_env() is ignored. Refer to the manpage for ap_env() for the attribute default values.
- ap_set_env() is used to change a value of the specified attribute. To change a list of attributes, you must use multiple ap_set_env() calls (one for each attribute changed)
- ap_get_env() is used to read a value of a specified attribute.

When you use ap_get_env() for attributes which are stored as data structures, memory is allocated for you. When you have finished using the information from the data structure, must use ap_free() to free the memory.

For memory allocated by your application, do not use ap_free(). Implement your own routines to free application-created memory allocations.

A partial list of A/P attributes used that require allocating and deallocating memory are: AP_PCDL, AP_PCDRL, and AP_LCL_PADDR.

Determine Initiator and Responder Roles

When you create your programs, decide which process will be initiating and which one will be accepting associations.

By default, both processes may initiate and respond to a request. The roles are set using the AP_ROLE_ALLOWED environment attribute.

However, when one process issues a connection request, it is established as the initiator and may no longer receive connection indications. The AP_ROLE_CURRENT is set to AP_INITIATOR.

On the other hand, when the connection indication is received by the other process, it is established as the responder, and can no longer issue a connection request. The AP_ROLE_CURRENT is automatically set to AP_RESPONDER.

If you decide to have a process act as a responder or initiator, you can set the AP_ROLE_ALLOWED attribute using the A/P environment call:

```
ap_set_env(fd,AP_ROLE_ALLOWED,val)
```

Programs can be managed by designating the paired programs as Initiator and Responder. See manpages for example programs.

Setting AP_ROLE_ALLOWED only affects the ability to send and receive a connection request. Other primitives, such as P_DATA_IND, are not controlled by the AP_ROLE_ALLOWED attribute.

CAUTION The initiator and responder roles must be established before binding the communication endpoint to the local presentation address (in the AP_UNBOUND state). Any change in AP_ROLE_ALLOWED after binding is ignored.

Determine Address Handling

Before a communication endpoint can be used in an ACSE/Presentation application, a unique presentation address must be assigned (or bound) to it.

The presentation address consists of a presentation selector, session selector, transport selector, and a n optional network service access point (NSAP).

APRI also supports the binding of the local presentation address without specifying a network address. This allows a single responder to receive association indications for any of the subnetworks configured on an HP OTS/9000 system. To use this feature, when setting the local address through the AP_BIND_PADDR environment attribute, set the n_nsaps field of the ap_paddr_t structure to zero. When this feature is used, no local NSAP information need be provided in the ap_paddr_t structure.

Once a communication endpoint has been created by calling ap_open(), and the initiator and responder roles have been established or explicitly assigned using ap_set_env() with the AP_ROLE_ALLOWED primitive, the endpoint is bound to the presentation address using the A/P environment call

```
ap_set_env(fd,AP_BIND_PADDR,paddress)
```

Note that rebinding the presentation address is not supported in this release. Binding can only be done in the AP_UNBOUND state.

A/P supports extended addressing which provides for the maximum address lengths for each selector in the presentation address as shown in the following table:

Table 4-1 Selector Maximum Lengths

Selector	Bytes
Presentation	16
Session	16
Transport	32
NSAP	20

NOTE Other OSI services implementations (non-HP) may not support the same maximum length for the presentation selector. If your system needs to communicate with other OSI services, make sure to check the maximum size supported for the presentation selector.

The NSAP cannot be NULL. NSAPs used for ACSE/Presentation applications must be configured by the network administrator. T he NSAP must be unique for the system. The presentation address used by your ACSE/Presentation application must be different than any other presentation addresses used by other application services on your system if you plan to operate these applications at the same time.

For example, if the HP 9000 FTAM service is being used, the address you create for A/P applications must be different than the FTAM addresses. See the *Installing, and Administering the OSI Transport Services/9000* manual for more information o n presentation addresses.

Negotiating Application and Presentation Contexts

Both application contexts (specified using the AP_CNTX_NAME attribute) and presentation contexts (AP_PCDL attribute) must be negotiated by A/P associations.

Application Contexts (AP_CNTX_NAME)

Associations must negotiate the application contexts used. The initiator proposes a service or services to run, and the responder must reply to accept or reject the context. This is a mandatory parameter in the OSI ACSE layer. For more information, refer to the OSI ACSE standard (ISO 8649, 8650).

Presentation Contexts (AP_PCDL)

The presentation context definition list (PCDL) is a list of presentation contexts. Each presentation context consists of a unique presentation context id, an abstract syntax and one or more transfer syntaxes.

Processes must negotiate the PCDLs used. The initiator proposes contexts using the AP_PCDL attribute. The responder returns the result for each proposed item with the status of ACCPT, USER_REJ or PROV_REJ in the AP_PCDRL attribute before sending the A_ASSOC_RSP primitive. An application can use ap_get_env() on AP_PCDRL to find out the status and reason for each proposed item.

After a connection has been established, AP_DCS contains all acceptable presentation contexts by both the initiator and responder sides.

Portability, Migration Considerations

To ease the migration of your applications as the standards evolve, the following items should be considered:

- Set the AP_LIB_SEL attribute before any other attributes. This attribute indicates which version of the A/P library will be used for the process. This implementation currently only supports one. However, as the A/P specification evolves, others may be supported.
- In future releases, the A/P interface will be modified to reflect the X/OPEN ACSE/Presentation API Interface (XAP) once the standard is approved and finalized. If you expect to migrate to XAP, the following coding guidelines can help you:
  - Always include the A/P library: ap_lib.h in your code that uses ap_xxx() calls and A/P data structures.
  - Isolate and minimize the code using ap_xxx() calls and A/P data structures.

- Isolate the allocations and freeing of memory passed to the A/P library to a single function. For example:

  ```
  void *ubuf_ptr;
  user_get_osi_buf (&ubuf_ptr, len,”data to put in ubuf”);
  ap_snd (.., ubuf_ptr);
  ```

- Do not statically initialize a cdata structure. For example, do not use the statement:

  ```
  a_assoc_req_env_t peer_application = {...};
  ```

- Do not use the udata_length part of the cdata parameter for inbound indications. Instead, calculate the length from the osi_buf that is returned. Isolate this calculation to one routine.

3. Managing Data

ACSE/Presentation (A/P) defines two types of data: control data and user data. Control data (cdata) is associated with the primitives and its content is specific to each primitive. User data (ubuf) is managed by the application and is defined by the context and transfer syntax selected.

Control Data

Control data is associated with the supported ACSE/Presentation primitives and is managed by the A/P library. Control data is stored in the cdata structure and consists of additional protocol information that must be conveyed to the receiving process using the ap_snd() and ap_rcv() calls. An example of a primitive that contains the cdata structure is A_ASSOC_REQ which passes the proposed initial token assignment to the responder.

For each A/P primitive, the specific information required for the primitive is saved in its own cdata structure.

User Data

User data is managed by the application. User data must be encoded and decoded by the application. The two applications or processes in an association must agree to the presentation context used as previously discussed. Refer to the ACSE/Presentation interface manpages for information on encoding each primitive.

Multiple Sends and Receives of Data

If you decide to send a single primitive using multiple ap_snd() calls, or the user data is too large to send in one buffer, you can set the AP_MORE bit in the flags parameter to help assure complete receipt of the data by the other process. Each additional ap_snd() call must use the same sptype. The last ap_snd() call must reset the AP_MORE bit to indicate the end of the data. For example:

1) First ap_snd()	set AP_MORE, and send data
2) Additional ap_snd()'s	set AP_MORE, and send data
3) Last ap_snd()	re-set AP_MORE, and send last data

When receiving data, the AP_MORE bit is set by the A/P library if the receiving ubuf is not large enough, or if a partial primitive was received. In either case, the application must continue to invoke ap_rcv() to receive the remainder of the data.

It is recommended that after each ap_rcv() call, the sptype is checked. Some primitives such as A_ABORT_IND are not flow-controlled and may interrupt the receipt of a partially received primitive. In this case, the remaining data from the previous primitive is lost.

User Data Encoding/Decoding

The ACSE/Presentation library is not responsible for user data encoding/ decoding. The application programmer must choose an ASN.1 compiler and run-time library. Refer to the ASN.1 compiler documentation for the product you purchased. The sample programs in the manpages use the Marben ASN.1 compiler product and run-time library.

4. Using Synchronous vs. Asynchronous Mode

ACSE/Presentation supports both synchronous and asynchronous mode execution of calls. In synchronous mode, an ACSE/Presentation call blocks until the call can be completed. While blocked, no other tasks can be performed. Synchronous mode is the default.

In asynchronous mode, an ACSE/Presentation call is not blocked (except under kernel resource shortages). The function completes as much of its task as it can, then returns to the user.

Synchronous Mode

Synchronous mode is useful for processes maintaining a single connection, and for processes that can wait for events to occur.

When synchronous mode is used, ap_snd() blocks until resources are available to send the entire primitive. Similarly, ap_rcv() blocks until either an entire primitive is received, or the user buffer is filled by the library. In the latter case, ap_rcv() returns with the AP_MORE bit set in the flags argument. To receive the remaining data, the application must continue to call ap_rcv() until it returns with the AP_MORE bit reset.

Asynchronous Mode

Asynchronous mode is used for applications in which long delays are expected between events, and for processes that can perform other tasks while waiting for events to happen. Asynchronous mode is also useful for managing multiple connections concurrently.

Asynchronous mode is specified in ap_open() by setting the O_NDELAY bit in the oflags parameter. When used asynchronously, ap_snd() and ap_rcv() do not block in most cases.

If there are insufficient resources from OTS to send the entire primitive for an ap_snd(), a partial send can occur. A return code of -1 is received and ap_errno is set to AP_AGAIN. To complete sending the primitive, ap_snd() must be invoked again with the same set of buffers and arguments until the call returns successfully.

If ap_rcv() is called in asynchronous mode, data is read from the communication endpoint until either:

- the entire primitive is received
- the ubuf argument buffer is full
- no more user data is available.

In the second case, the AP_MORE bit is set in the flags parameter of ap_rcv(). If the AP_MORE bit is set, a primitive was partially received. To receive the rest of the primitive, the ap_rcv() must be re-invoked until the call returns successfully.

In the third case:

- Either ap_rcv() returns successfully with the AP_MORE bit set and ubuf is partially filled. In this case, to use the remaining ubuf space for the next ap_rcv() you must re-adjust ubuf->len.
- Or, ap_rcv() returns unsuccessfully with ap_errno set to AP_AGAIN. In this case, you must not alter any parameters for the next ap_rcv(). Refer to the discussion in the ap_rcv() manpage for more information.

Check the sptype after each ap_rcv() call. Some primitives such as A_ABORT_IND are not flow-controlled and may interrupt the receipt of a partially received primitive. In this case, the remaining data of the original primitive is lost.

Note that ap_rcv() may still be blocked by running in asynchronous mode in memory shortage situations such as when using large numbers of connections.

Changing Modes

Once a communication endpoint has been opened using synchronous mode, it can be changed to asynchronous mode in the application by setting the AP_STREAM_FLAGS attribute with AP_NDELEY.

However, to reset the execution to synchronous mode, the communication endpoint must be closed and reopened without using O_NDELEY in the ap_open() call.

5. Managing Multiple Connections

Before writing an application with multiple connections, you may need to consider the following items:

- Execution mode
- Connection retry
- Resource constraints

Execution Mode

For using multiple connections, asynchronous mode is recommended. Using multiple connections introduces factors that cannot be controlled such as: when a task will complete, how long a task will take, or whether or not tasks will complete sequentially. Factors affecting task completion include network traffic and the load on the system. Using synchronous mode can introduce deadlock situations. For example, consider the following scenario using first synchronous then asynchronous processing:

Two processes, an initiator and responder are each managing two communication endpoints. The initiator side of the communication endpoints are labeled A and B. The responder side of the communication endpoints are labeled C and D (see Figure 4-1).

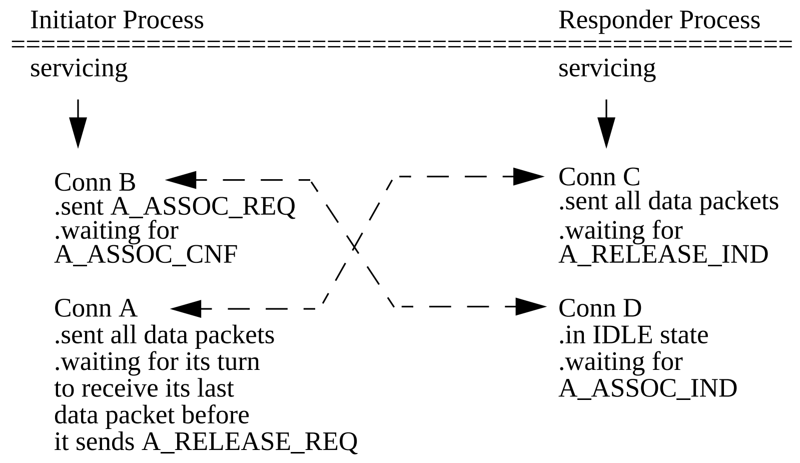
Communication endpoint A initiates a connection and transfers two data packets over the connection to C. Two data packets are transferred over the connection from C to A. Then endpoint A initiates a connect release after it receives the two data packets.

Now, since the completion time of each task is not guaranteed (as previously described), the following situation could occur:

- The initiator process is servicing endpoint B which has just sent out a connection request (A_ASSOC_REQ) and is waiting for confirmation (A_ASSOC_CNF).
- Endpoint A received the first data packet and is waiting for its turn to receive the last data packet already on the queue before it sends out a release request (A_RELEASE_REQ). At the same time, the responder process is servicing endpoint C which has already sent its two data

packets and is waiting for the release indication (A_RELEASE_IND) while endpoint D is in AP_IDLE state waiting for the connection indication (A_ASSOC_IND).

Figure 4-1 Multiple Connections Scenario



- If the connections are using synchronous mode, a deadlock situation occurs. Endpoint A is blocked until endpoint B receives a A_ASSOC_CNF which will never happen, because the remote process is servicing endpoint C which is waiting for A_RELEASE_REQ sent by endpoint A and endpoint A is blocked.
- On the other hand, using asynchronous mode, ap_rcv() will return AP_AGAIN on communication endpoint B because there is no data to process. It would not block the process, and the application can continue to service the next connection. Communication endpoint A would then be able to receive its last data packet and send a release request (A_RELEASE_REQ) to endpoint C. Then C would be able to process its release indication (A_RELEASE_IND) and reply (A_RELEASE_RSP). Endpoint D would receive its connection indication (A_ASSOC_IND) and so on.

For more information on how an application can handle synchronous and asynchronous mode processing, see the section “Using Synchronous vs. Asynchronous Mode.”.

Connection Retry

When there is a resource shortage, ap_snd() may return AP_AGAIN. The application should re-send the same primitive as outlined in the ap_snd() manpage.

NOTE In the case of a connection request (A_ASSOC_REQ), if a resource shortage occurs in the OTS stack, a reject may be received as a A_PABORT_IND or A_ABORT_IND.

A_PABORT_IND indicates a shortage of memory in the OTS stack while trying to establish a connection. This is caused by the dynamic memory allocation scheme in the stack. A_ABORT_IND may occur if there is a memory shortage on the remote side of the connection request.

To handle this situation, an application can implement a connection retry mechanism to re-send the request again. Keep in mind that this type of resource shortage problem should be temporary. To avoid waiting indefinitely, set a counter to keep track of the number of retries.

Resource Constraints

Each connection will take up some resources of the system. These resources may include file descriptors and memory space. Since A/P uses ap_open() to create a communication endpoint and each endpoint corresponds to a file descriptor, there is a limited number of file descriptors that can be opened per process. For memory usage, when it is initially started, each A/P instance will take up about 540 bytes. This memory is consumed by the storage of the A/P environment attributes and buffer space for the A/P library itself.

The actual size of memory used per connection depends on the values of its environment attributes. For example, AP_BIND_PADDR may have a longer or shorter address. In addition to the A/P library memory usage, connections use some space within the OTS stack at each layer including the presentation, session, and transport layers.

For an application with a large number of connections, you need to be aware of the following system behavior and limitations:

- System maximums
- Process maximums
- Other application interactions

Note that if an ACSE/Presentation call (such as ap_open()) fails because of temporary memory shortage, the application may want to retry the same call later.

System Maximums

OTS supports up to 4096 virtual circuit (VC) connections at the transport and network layer (X.25 CONS). If applications such as X.400 or FTAM are also in use, note that each session access connection also uses a transport connection. If XTI API applications are also in use, note that each XTI connection also uses a transport connection. The number of X.25 VCs supported may be further limited by hardware configuration limits and the number and type of X.25 interface cards.

The system has no priority for connection requests versus data transfers on existing connections. It is recommended that an application successfully bring up all required connections before transferring data.

Connection management should not occur during data transfer. The application pacing should avoid possible timing problems.

Process Maximums

It is possible to have a single process use up to the maximum number of connections. Under these circumstances, observe the following restrictions:

- A low rate of incoming data, including connect indications.
- Incoming events do not occur at the same time on all of the connections. Too many concurrent events can overflow buffers. The AP_AGAIN error may occur more often.
- The application should receive incoming events as soon as possible to regularly poll the connection for events.
- Check burst rate.
- Check pacing of execution watching for timing errors.
- Check if the number of connections gradually increases or decreases.

Because communication endpoints are file descriptors, the maximum number of communication endpoints (and connections) per process is limited by the maximum number of open files allowed per process by HP-UX. The number of connections allowed per process is further limited by any other files opened by the process.

6. Other Tasks

Other items to consider in developing A/P applications include:

- File descriptors
- Signal handling
- Error checking

File Descriptors

Communication endpoints for ACSE/Presentation applications are HP-UX file descriptors. Because of unpredictable results, HP recommends you do not use HP-UX file system calls such as exec(2), dup(2), read(2), write(2), ioctl(2), or select(2) with HP OTS/9000 A CSE/Presentation file descriptors. Note that fork(2) can be used, but do not use fork(2) with exec(2).

A communication endpoint is returned when the ACSE/Presentation call ap_open() is invoked.

Note that the user may need to increase the open file limit for the application process to support a large number of connections. Refer to the getrlimit() and setrlimit() system call manpages for more information on changing file limits.

Signal Handlers

ACSE/Presentation calls interrupted by the arrival of a signal will not be restarted by the library. You are responsible for managing signals and providing recovery routines for the duration of any ACSE/Presentation calls. Signals are interrupts such as when you enter ˆC on a terminal to exit from a program.

If an interrupt occurs while executing A/P functions, the operating system class error EINTR is returned. When detected, you must re-invoke the A/P function. If EINTR is detected while executing ap_snd() or ap_rcv(), the call must be invoked again as described in how to handle AP_AGAIN in the ap_snd() and ap_rcv() manpages (for both synchronous and asynchronous processing). The A/P library continues the call from where it left off.

Signal handlers can be written to capture signals and exit, or to prevent signals from interrupting critical call sequences.

Error Checking Routines

Along with processing signals, ACSE/Presentation applications need to check for return codes from calls and particular error conditions that are returned. Error conditions related to ACSE/Presentation as well as other protocol errors can be returned.

The global variable ap_errno and the ap_error() function are used for error reporting. The ap_error() call prints an ASCII string error message corresponding to the last received ap_errno. A/P library errors and system errors can be received in the ap_errno variable.

A list of A/P specific errors you can access using ap_error() is included in the ap_intro manpage.

For determining where problems occur, you can also use API tracing which is described in Chapter 5,Ttroubleshooting your Application.

Error-handling in Multi-Threaded Applications

Multi-threaded applications must define ap_errno as:

```
extern unsigned long _ap_errno()
#define ap_errno _ap_errno()
```

This ensures that the application can still use ap_errno which would return the thread-specific ap_errno value to the application.

7. Before Running ACSE/Presentation Applications

In order to run ACSE/Presentation applications between two processes or systems, make sure:

- your local and remote systems have the correct configuration such as local and destination NSAPs
- that the OTS stack has been started successfully on both your local and remote systems
- you start your responder process first.

Multi-threaded ACSE/Presentation Example

Multi-threaded sample programs with similar functionality are also included online in the `/opt/ots/apli/demo/threads_demo` directory. These programs use Kernel Threads.

ACSE/Presentation Example

Sample programs are included online in the /opt/ots/apli/demo directory. You'll also find:

- A readme file describing how the programs work.
- Header files included by the programs in the `incl` subdirectory.
- Functions specific to the Marben ASN.1 compiler and run-time library to build, encode, decode, and free PDUs or C data structures generated by the compiler are included in the file: asn1_if.c.

NOTE Note: To use a different compiler, customize these functions for the C data structures and encoding/decoding routines generated by your ASN.1 compiler.

- Main program for the initiator process: init.c
- Main program for the responder process: resp.c
- Utility program used by both the initiator and responder: utils.c

Program using ROSE

Additional programs are included online in `/opt/ots/rose/demo` which exercise the ROSE library functions and provide an example using ROSE API tracing. Multi-threaded sample programs with similar functionality are also included online in the `/opt/ots/rose/demo/threads_demo` directory. These programs use Kernel Threads. Refer to the online README f ile for more information.

5 Troubleshooting Your Application

This chapter describes API tracing for ACSE/Presentation followed by API tracing for ROSE.

Using A/P API Tracing

You enable tracing in your application program and the tracing output is printed to a trace file. Different levels of tracing are provided with the trace facility. For example, you can trace procedure entry and exits, error conditions, or both.

A/P API Tracing Using Environmental Variables

A/P API has been enhanced to allow control of tracing via environment variables. The original API tracing mechanism provided through global variables is still available, however the new method means you do not need to write any special code in your application to take advantage of API tracing.

The following is an example of the new mechanism for APRI:

```
export AP_TRACE=io
initiator -nlocal_node -Nremote_node

Thu May 16 10:21:46 1996 -> ap_open()
  > pathname = /dev/osipi
  > oflags = 0x0
Thu May 16 10:21:47 1996 <- ap_open() = 4

Thu May 16 10:21:47 1996 -> ap_init_env()
  fd = 4.
  env_file = NULL
  flags = 0.
Thu May 16 10:21:47 1996 <- ap_init_env() = 0

Thu May 16 10:21:47 1996 -> ap_set_env()
  fd = 4.
  attr = AP_ROLE_ALLOWED
  *val = 1
Thu May 16 10:21:47 1996 <- ap_set_env() = 0
```

This example shows the environment variable, AP_TRACE, being set to the ASCII characters “io” (meaning input and output parameters). For csh(1) users the setenv(1) command should be used. The program is then invoked and the API tracing is automatically enabled to the level specified in the environment variable.

The program shown does not require any special user code for handling the environment variables, or controlling the API tracing global variables. The mechanism works by having the AP library itself test the environment variables on the first valid APRI call (always ap_open) and then set the global API trace variables according to the value of the environment variables. If no environment variables are present, then no API tracing takes place.

Environment Variable Names

The names of the ACSE/Presentation (AP) environment variables available are as follows:

AP_TRACE

AP_TRACE_FILE

AP_TRACE_MAX_UDATA

See “Trace Output” for the effects of these environment variables.

AP_TRACE:. The value defined for this variable indicates the trace level to be used for each API call. It is actually a set of flags, defined in the file /usr/ include/api_trace.h.

Default = 0 (trace_off).

You may set this value in one of two ways:

1. It may be treated as an integer value and may be set using either decimal or hexadecimal notation, for example:

   ```
   AP_TRACE=0xff
   ```

2. It may be treated as a set of ASCII flags. The following standard flags are defined.

   i = input parameters

   o = output parameters

   x = external procedure entry exit

   e = error tracing

   n = internal tracing

AP_TRACE_FILE:. The name of the file that is to receive tracing results.

Default = stderr.

AP_TRACE_MAX_UDATA:. The maximum amount of user data (in bytes) that will be displayed when parameters are displayed.

Default = 16.

Tracing Using Global Variables

A/P API tracing is controlled by three global variables. The variables are described below:

ap_trace	An integer value which constitutes a bitmask to control the level of tracing performed. By default this mask is 0.
ap_trace_fp	A pointer to a UNIX file to receive the tracing output. By default this is set to stderr.
ap_trace_max_udata	The maximum amount of user data (in bytes) that will be displayed during tracing. The default is 16.

Selecting Types of Tracing

The level of A/P tracing is controlled by the ap_trace variable.

The ap_trace variable is defined as a bitmask that can be set to particular values (as defined in the file /opt/ots/lib/api_trace.h) and as listed in the following table.

For example, if you want to trace output parameters only, you would set ap_trace to API_TR_OUTPUT.

API_TR_ENTRY_EXIT	Traces procedure entry and exit. No parameter information is displayed. This is useful if you are only interested in seeing what A/P calls your program is making. Note that this trace is automatically generated if you use API_TR_INPUT, API_TR_OUTPUT, or API_TR_INT_ENTRY_EXIT.
API_TR_INPUT	Provides traces of A/P function call input parameters. This is useful if you want to verify that A/P is actually receiving the values you expect.
API_TR_OUTPUT	Provides traces of A/P function call output parameters. This is useful if you want to verify what values A/P is passing back to your program.
API_TR_INT_ENTRY_EXIT	Enables internal tracing. Use if directed to do so by your HP support representative.
API_TR_INT_ERROR	Enables external and internal error tracing. Recommended that users turn this tracing on at all times. Includes filename and line number information useful for HP factory support.

Enable A/P API Tracing Using Global Variables

To enable A/P tracing in your program, add the following statements to your program:

1. Include the appropriate definitions by adding these lines:

```
#include <stdio.h>
#include <api_trace.h>
extern int ap_trace;
extern int ap_trace_max_udata;
extern FILE *ap_trace_fp;
```

2. Within your program, enable tracing and select the level of tracing you want by modifying the value of the ap_trace variable. For example, to enable procedure tracing, input parameters tracing, output parameters tracing, and error tracing enter the statement:

   ```
   ap_trace = API_TR_ENTRY_EXIT | API_TR_INPUT |
   API_TR_OUTPUT|API_TR_INT_ERROR;
   ```

3. If you want to redirect the trace output from the default file stderr enter the statement: where /tmp/my_ap_trace is the name of the file you choose for tracing.

   ```
   if ((ap_trace_fp = fopen(“/tmp/my_ap_trace”, “w”)) ==
   NULL) ap_trace_fp = stderr;
   ```

4. If you want more than the first 16 bytes of data to be displayed (the default), then modify the ap_trace_max_udata parameter. For example to increase the data displayed to 256 bytes enter:

   ```
   p_trace_max_udata = 256;
   ```

Trace Output

The format of the trace output is as follows:

• The first line indicates what function is being called and at what time. The “->” symbol indicates procedure entry for example:

  ```
  15:53:38 -> ap_open()
  ```

• The subsequent line(s) indicate the input parameters to the function call. The parameter name and its value are listed. For example:

  ```
  14:53:39 -> ap_set_env()
     fd = 3.
     attr = AP_ROLE_ALLOWED
   *val = 1
  ```

• After the input parameters are displayed, the procedure exit is shown. The time of return, and the return value are shown. The “<-” symbol indicates procedure exit. For example:

  ```
  15:53:38 <- ap_open() = 3
  ```

• Lastly, any output parameters returned by the function call are displayed. The parameter name is listed with its value. For example:

```
15:53:38 <- ap_set_env() = 0
    attr = AP_CNTX_NAME
    val->len = 4
    val->buf = (4/4)
    52 01 00 04 R....
```

- The amount of data displayed is based on the value set for ap_trace_max_udata. For example, if ap_trace_max_udata is set to 16, only the first 16 bytes are displayed as shown below:

  ```
  val->sad.buf = (16/22)
  06 69 5F 73 73 65 6C 06 69 5F 74 73 65 6C 07
  48.i_ssel.i_tsel.H
  ```

  The item "(16/22)" indicates that 16 of the 22 bytes of data output is displayed. The following line contains the actual data in both hexadecimal and ASCII representation.

Note that for some of the A/P calls such as ap_snd(), ap_rcv() and ap_poll() the notation ">" is used to indicate input parameters and "<" indicates output parameters. For example:

```
14:53:40 -> ap_poll()
> nfds = 1
> timeout = -1
> fds = 0x68FAC6A8
> fds[0].fd = 3
> fds[0].events = 0x4
< fds[0].revents = 0x4
14:53:40 <- ap_poll() = 1
```

Tracing in Multi-threaded Applications

A/P API traces in multi-threaded applications include thread-ids of the executing threads as part of the trace output. This identifies a particular part of the trace output as belonging to a particular thread. The format of the trace output for multi-threaded applications is as follows:

```
>>> thread-id  = 5 >>>
Wed Mar  5 15:26:30 1997 -> ap_open()
<<< thread-id  = 5 <<<

>>> thread-id  = 6 >>>
Wed Mar  5 15:26:30 1997 -> ap_open()
<<< thread-id  = 6 <<<

>>> thread-id  = 7 >>>
Wed Mar  5 15:26:30 1997 -> ap_open()
<<< thread-id  = 7 <<<
>>> thread-id  = 5 >>>
>  pathname = /dev/osipi
>  oflags = 0x0
<<< thread-id  = 5 <<<
```

```
>>> thread-id  = 6 >>>
>  pathname = /dev/osipi
>  oflags = 0x0
<<< thread-id  = 6 <<<
>>> thread-id  = 7 >>>
>  pathname = /dev/osipi
>  oflags = 0x0
<<< thread-id  = 7 <<<
```

The line >>> *thread-id = 5* >>> denotes that the lines following it have been output by the thread with the thread-id 5. Also, the line <<< *thread-id = 5* <<< denotes the end of trace output from the thread with thread-id 5. So, every group of tracing lines is enclosed within lines of the form >>> *thread-id = no* >>> and <<< *thread-id = no* <<<. This unambiguously identifies the thread which has output these tracing lines. The tracing output from various threads may be interleaved which denotes that they are being scheduled in and out while they are executing and doing tracing output.

Using ROSE API Tracing

Tracing is enabled in your application program and the tracing output is printed to a trace file. Different levels of tracing are provided with the trace facility. For example, you can trace procedure entry and exits, error conditions, or both. ROSE tracing has been implemented to be consistent with ACSE/Presentation tracing.

Environment Variable Names

The names of the ROSE environment variables available are as follows:

ROSE_TRACE

ROSE_TRACE_FILE

ROSE_TRACE_MAX_UDATA

The effect of these environment variables below.

AP_TRACE:

These values indicates the trace level to be used for each API call. It is actually a set of flags, defined in the file //opt/ots/lib/api_trace.h.

Default = 0 (trace_off).

You may set this value in one of two ways:

1. It may be treated as an integer value and may be set using either decimal or hexadecimal notation, for example:

   ```
   AP_TRACE=0xff
   ```

2. It may be treated as a set of ASCII flags. The following standard flags are defined.

   i = input parameters

   o = output parameters

   x = external procedure entry exit

   e = error tracing

   n = internal tracing

AP_TRACE_FILE:

The name of the file that is to receive tracing results.

Default = stderr.

AP_TRACE_MAX_UDATA:

The maximum amount of user data (in bytes) that will be displayed when parameters are displayed.

Default = 16.

Tracing Using Global Variables

ROSE API tracing is controlled by three global variables. The variables are described below:

Table 5-1 **ROSE Trace Variables**

rose_trace	An integer value which constitutes a bitmask to control the level of tracing performed. By default this mask is 0
rose_trace_fp	A pointer to a UNIX file to receive the tracing output. By default this is set to stderr.
rose_trace_max_udata	The maximum amount of user data (in bytes) that will be displayed during tracing. The default is 16.

Selecting Types of Tracing

The level of ROSE tracing is controlled by the rose_trace variable.

The rose_trace variable is defined as a bitmask that can be set to particular values (as defined in the file /opt/ots/lib/api_trace.h) and as listed in Table 5-1.

For example, if you want to trace output parameters only, you would set rose_trace to API_TR_OUTPUT.

Table 5-2 ROSE Tracing Types

API_TR_ENTRY_EXIT	Traces procedure entry and exit. No parameter information is displayed. This is useful if you are only interested in seeing what ROSE calls your program is making. Note that this trace is automatically generated if you use API_TR_INPUT, API_TR_OUTPUT, or API_TR_INT_ENTRY_EXIT.
API_TR_INPUT	Provides traces of ROSE function call input parameters. This is useful if you want to verify that ROSE is actually receiving the values you expect.
API_TR_OUTPUT	Provides traces of ROSE function call output parameters. This is useful if you want to verify what values ROSE is passing back to your program.
API_TR_INT_ENTRY_EXIT	Enables internal tracing. Use if directed to do so by your HP support representative.
API_TR_INT_ERROR	Enables external and internal error tracing. Recommended that users turn this tracing on at all times. Includes filename and line number information useful for HP factory support.

Enable ROSE API Tracing

To enable ROSE tracing in your program, add the following statements to your program:

1. Include the appropriate definitions by adding these lines:

```
#include <stdio.h>
#include <api_trace.h>
extern int rose_trace;
extern int rose_trace_max_udata;
extern FILE *rose_trace_fp;
```

2. Within your program, enable tracing and select the level of tracing you want by modifying the value of the rose_trace variable. For example, to enable procedure tracing, input parameters tracing, output parameters tracing and error tracing, enter the statement:

   ```
   rose_trace =
   API_TR_ENTRY_EXIT|API_TR_INPUT|API_TR_OUTPUT|API_TR_I
    NT_ERROR;
   ```

3. If you want to redirect the trace output from the default file stderr enter the statement:

   ```
   if ((rose_trace_fp=fopen(“/tmp/my_ro_trace”, “w”))
   == NULL); rose_trace_fp = stderr;
   ```

   where /tmp/my_ro_trace is the name of the file you choose for tracing.

4. If you want more than the first 16 bytes of data to be displayed (the default), then modify the rose_trace_max_udata parameter. For example to increase the data displayed to 256 bytes enter:

   ```
   rose_trace_max_udata = 256;
   ```

Trace Output

ROSE tracing is included in the online ROSE program example. The format of the trace output is consistent with the format described for A/P tracing:

- The first line indicates what function is being called and at what time. The “->” symbol indicates procedure entry. For example:

  ```
  12:54:30 -> ro_bind()
  ```

- The subsequent line(s) indicate the input parameters to the function call. The parameter name and its value are listed.
- After the input parameters are displayed, the procedure exit is shown. The time of return, and the return value are shown. The “<-” symbol indicates procedure exit. For example:

  ```
  12:54:30 <- ro_bind() NO ERRORS
  ```

- Lastly, any output parameters returned by the function call are displayed. The parameter name is listed with its value.

- The amount of data displayed is based on the value set for rose_trace_max_udata. For example, if rose_trace_max_udata is set to 16, only the first 16 bytes are displayed.

Tracing in Multi-threaded Applications

The format of the trace output in ROSE API traces in multi-threaded applications is consistent with the format described for A/P API traces in similar applications.

A ACSE/Presentation Reference Pages

This appendix includes a list of the supported ACSE/Presentation attributes and a glossary of terms

ACSE/Presentation Primitives

Table A-1 AP_ENV Table of Attributes

Name of attribute/ Purpose	Data Type	Default value of attribute	Values legal for this attribute (* = not supported by current release)	Readable states (value of AP_STATE attribute)	Writable states (value of AP_STATE attribute)
`AP_ACSE_AVAIL` Bitmask indicating available versions of the ISO ACSE protocol	unsigned long	`or'd bits:` `AP_ACSEVER1`	`bit values:` `AP_ACSEVER1`	always	never
`AP_ACSE_SEL` Bitmask indicating which version of the ISO ACSE protocol is currently selected	unsigned long	or'd bits: `AP_ACSEVER1`	bit values: `AP_ACSEVER1`	always	only in states: `AP_UNBOUND` `AP_IDLE`
`AP_BIND_PADDR` Presentation address to which the stream supporting this instance of the A/P-Library is bound	ap _paddr_t	N.A.	Any allowed by the C type ap_paddr_t	always	only in state `AP_UNBOUND`
`AP_CLD_AEID` Called application entity invocation identifier	long	`AP_CLD_AEID_NOVAL`	Any allowed by the C type long	always	only in states: `AP_UNBOUND` `AP_IDLE`

Name of attribute/ Purpose	Data Type	Default value of attribute	Values legal for this attribute (* = not supported by current release)	Readable states (value of AP_STATE attribute)	Writable states (value of AP_STATE attribute)

`AP_CLD_AEQ`
Called application entity qualifier
	any_t	None	Any allowed by the C type any_t	always	only in states:
					`AP_UNBOUND`
					`AP_IDLE`

`AP_CLD_APID`
Called application process invocation identifier
	long	`AP_CLD_APID_NOVAL`	Any allowed by the C type long	always	only in states:
					`AP_UNBOUND`
					`AP_IDLE`

`AP_CLD_APT`
Called application process title
	any_t	None	Any allowed by the C type any_t	always	only in states:
					`AP_UNBOUND`
					`AP_IDLE`

`AP_CLD_CONN_ID`
Session connection identifier proposed by the association-responder
	ap_conn _id_t	None	Any allowed by the C type ap_conn_id_t	always	only in states
					`AP_UNBOUND`
					`AP_IDLE`
					`AP_WASSOCrsp_`
					`ASS OCind`

`AP_CLG_AEID`
Calling application entity invocation identifier
	long	`AP_CLG_AEID_NOVAL`	Any allowed by the C type long	always	only in states:
					`AP_UNBOUND`
					`AP_IDLE`

`AP_CLG_AEQ`
Calling application entity qualifier
	any_t	None	Any allowed by the C type any_t	always	only in states:
					`AP_UNBOUND`
					`AP_IDLE`

Name of attribute/ Purpose	Data Type	Default value of attribute	Values legal for this attribute (* = not supported by current release)	Readable states (value of AP_STATE attribute)	Writable states (value of AP_STATE attribute)
AP_CLG_APID Calling application process invocation identifier	long	AP_CLG_APID_NOVAL	Any allowed by the C type long	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_APT Calling application process title	any_t	None	Any allowed by the C type any_t	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_CONN_ID Session connection identifier proposed by the association-initiator	ap_conn _id_t	None	Any allowed by the C type ap_conn_id_t	always	only in states: AP_UNBOUND AP_IDLE
AP_CNTX_NAME Application context for the association	objid_t	N.A.	Any allowed by the C type objid_t	always	only in states AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASS OCind
AP_DCS Defined context set (This attribute consolidates information conveyed by the AP_PCDL and AP_PCDRL attributes.)	ap_dcs_t	N.A.	Any allowed by the C type ap_dcs_t	in any state but AP_UNBOUND	never

Name of attribute/ Purpose	**Data Type**	**Default value of attribute**	**Values legal for this attribute (* = not supported by current release)**	**Readable states (value of AP_STATE attribute)**	**Writable states (value of AP_STATE attribute)**
`AP_DPCN` Default presentation context name	ap_dcn_t	None	Any allowed by the C type ap_dcn_t *	never	never
`AP_DPCR` Default presentation context result	long	`AP_DPCR_NOVAL`	one of: `AP_DPCR_NOVAL` `AP_ACCEPT *` `AP_USER_REJ *` `AP_PROV_REJ *`	never	never
`AP_INIT_SYNC_PT` Initial synchronization point serial number	unsigned long	`AP_MIN_SYNCP`	range from `AP_MIN_SYNCP to AP_MAX_SYNCP(999999)`	always	only in states `AP_UNBOUND` `AP_IDLE` `AP_WASSOCrsp_ASS OCind`
`AP_LCL_PADDR` Presentation address used by the local entity	ap _paddr_t	N.A.	Any allowed by the C type ap_paddr_t	always	never
`AP_LIB_AVAIL` Bitmask indicating available versions of the A/P-Library	unsigned long	`AP_LIBVER1`	bit values: `AP_LIBVER1`	always	never
`AP_LIB_SEL` Bitmask indicating which version of the A/ P-Library is currently selected	unsigned long	`AP_LIBVER1`	bit values: `AP_LIBVER1`	always	only in state: `AP_UNBOUND`

Name of attribute/ Purpose	Data Type	Default value of attribute	Values legal for this attribute (* = not supported by current release)	Readable states (value of AP_STATE attribute)	Writable states (value of AP_STATE attribute)
`AP_MODE_AVAIL` Specifies the modes in which the ACSE services may operate	unsigned long	or'd bits: `AP_NORMAL_MODE` `AP_X410_MODE`	bit values: `AP_NORMAL_MODE` `AP_X410_MODE`	always	never
`AP_MODE_SEL` Specifies the mode in which the ACSE services will operate for this association	unsigned long	or'd bits: `AP_NORMAL_MODE`	bit values: `AP_NORMAL_MODE` `AP_X410_MODE`	always	only in states: `AP_UNBOUND` `AP_IDLE`
`AP_MSTATE` State of AP_MORE bits (both send and receive)	unsigned long	or'd bits: NULL	bit values: `AP_SNDMORE AP_RCVMORE`	always	never
`AP_PCDL` Presentation context definition list	ap_cdl_t	None	Any allowed by the C type ap_cdl_t	only in states: `AP_IDLE` `AP_WASSOCrs` `p_ASSOCind`	only in state: `AP_IDLE`
`AP_PCDRL` Presentation context definition result list	ap _cdrl_t	None	Any allowed by the C type ap_cdrl_t	all states except: `AP_IDLE` `AP_UNBOUND`	only in state: `AP_WASSOCrsp_` `ASS OCind`
`AP_PFU_AVAIL` Bitmask indicating which Presentation Layer functional units are available	unsigned long	or'd bits: NULL	bit values: NULL	always	never

Name of attribute/ Purpose	Data Type	Default value of attribute	Values legal for this attribute (* = not supported by current release)	Readable states (value of AP_STATE attribute)	Writable states (value of AP_STATE attribute)
AP_PFU_SEL Bitmask indicating which Presentation Layer functional units are currently selected	unsigned long	or'd bits: NULL	bit values: NULL	always	only in states AP_UNBOUND AP_IDLE AP_WASSOCrsp_ ASS OCind
AP_PRES_AVAIL Bitmask indicating available versions of the ISO Presentation Layer protocol	unsigned long	or'd bits: AP_PRESVER1	bit values: AP_PRESVER1	always	never
AP_PRES_SEL Bitmask indicating which version of the ISO Presentation Layer protocol is currently selected	unsigned long	or'd bits: AP_PRESVER1	bit values: AP_PRESVER1	always	only in states: AP_UNBOUND AP_IDLE
AP_REM_PADDR Presentation address of remote entity	ap _paddr_t	N.A.	Any allowed by the C type ap_paddr_t	always	only in states: AP_UNBOUND AP_IDLE
AP_ROLE_ALLOWED Indicates which roles (association-initiator or association-responder) the library user may play	unsigned long	or'd bits: AP_INITIATOR AP_RESPONDER	bit values: AP_INITIATOR AP_RESPONDER	always	only in state AP_UNBOUND

Name of attribute/ Purpose	Data Type	Default value of attribute	Values legal for this attribute (* = not supported by current release)	Readable states (value of AP_STATE attribute)	Writable states (value of AP_STATE attribute)

`AP_ROLE_CURRENT`
Indicates whether the library user is the association-initiator or the association-responder for the current association
	unsigned long	None	bit values:
			`AP_INITIATOR`
			`AP_RESPONDER`
				in any states but:
				`AP_UNBOUND`
				`AP_IDLE`
					never

`AP_RSP_AEID`
Responding application entity invocation identifier
	long	`AP_RSP_AEID_NOVAL`	Any allowed by the C type long	always	only in states
					`AP_UNBOUND`
					`AP_IDLE`
					`AP_WASSOCrsp_`
					`ASS OCind`

`AP_RSP_AEQ`
Responding application entity qualifier
	any_t	None	Any allowed by the C type any_t	always	only in states
					`AP_UNBOUND`
					`AP_IDLE`
					`AP_WASSOCrsp_`
					`ASS OCind`

`AP_RSP_APID`
Responding application process invocation identifier
	long	`AP_RSP_APID_NOVAL`	Any allowed by the C type long	always	only in states
					`AP_UNBOUND`
					`AP_IDLE`
					`AP_WASSOCrsp_`
					`ASS OCind`

`AP_RSP_APT`
Responding application process title
	any_t	None	Any allowed by the C type any_t	always	only in states
					`AP_UNBOUND`
					`AP_IDLE`
					`AP_WASSOCrsp_`
					`ASS OCind`

Name of attribute/ Purpose	**Data Type**	**Default value of attribute**	**Values legal for this attribute (* = not supported by current release)**	**Readable states (value of AP_STATE attribute)**	**Writable states (value of AP_STATE attribute)**

`AP_SESS_AVAIL`
Bitmask indicating available versions of the ISO Session Layer protocol
	unsigned long	or'd bits:
		`AP_SESSVER1`
		`AP_SESSVER2`
			bit values:
			`AP_SESSVER1`
			`AP_SESSVER2`
				always	never

`AP_SESS_OPT_AVAIL`
Bitmask indicating whether certain optional capabilities of the session layer are supported by the underlying protocol provider
	unsigned long	or'd bit
		`AP_UCBC`
			bit values:
			`AP_UCBC`
			`AP_UCEC *`
				always	never

`AP_SESS_SEL`
Bitmask indicating which version of the ISO Session Layer protocol is currently selected
	unsigned long	or'd bits:
		`AP_SESSVER2`
			bit values:
			`AP_SESSVER1`
			`AP_SESSVER2`
				always	only in states
					`AP_UNBOUND`
					`AP_IDLE`
					`AP_WASSOCrsp_`
					`ASS OCind`

Name of attribute/ Purpose	Data Type	Default value of attribute	Values legal for this attribute (* = not supported by current release)	Readable states (value of AP_STATE attribute)	Writable states (value of AP_STATE attribute)
AP_SFU_AVAIL Bitmark indicating which Session Layer functional units are available	unsigned long	or'd bits: AP_SESS_DUPLEX AP_SESS_MINORSYNC AP_SESS_RESYNC AP_SESS_NEGREL	or'd bits: AP_SESS_HALFDUPLEX* AP_SESS_DUPLEX AP_SESS_XDATA * AP_SESS_MINORSYNC AP_SESS_MAJORSYNC * AP_SESS_RESYNC AP_SESS_ACTMGMT * AP_SESS_NEGREL AP_SESS_CDATA * AP_SESS_EXCEPT * AP_SESS_TDATA *	always	never
AP_SFU_SEL Bitmark indicating which Session Layer functional units are currently selected	unsigned long	or'd bits: AP_SESS_DUPLEX	bit values: AP_SESS_HALFDUPLEX * AP_SESS_DUPLEX AP_SESS_XDATA * AP_SESS_MINORSYNC AP_SESS_MAJORSYNC * AP_SESS_RESYNC AP_SESS_ACTMGMT * AP_SESS_NEGREL AP_SESS_CDATA * AP_SESS_EXCEPT * AP_SESS_TDATA *	always	only in states AP_UNBOUND AP_IDLE AP_WASSOCrsp_ ASS OCind

Name of attribute/ Purpose	Data Type	Default value of attribute	Values legal for this attribute (* = not supported by current release)	Readable states (value of AP_STATE attribute)	Writable states (value of AP_STATE attribute)
AP_STATE Current state of the A/P- Library Interface	unsigned long	AP_UNBOUND	one of: AP_UNBOUND AP_DATAXFER AP_WASSOCrsp_ASSOCind AP_WASSOCcnf_ASSOCreq AP_WRELrsp_RELind AP_WRELcnf_RELreq AP_WRESYNrsp_RESYNind AP_WRESYNcnf_RESYNreq AP_WRELrsp_RELind_ init AP_WRELcnf_RELreq_ rsp AP_WACTDrsp_ACTDind AP_WACTDcnf_ACTDreq AP_WACTErsp_ACTEind AP_WACTEcnf_ACTEreq AP_WACTIrsp_ACTIind AP_WACTIcnf_ACTIreq AP_WSYNCMArsp_SYNCMA ind AP_WSYNCMAcnf_SYNCMA req AP_WCDATArsp_CDATAind AP_WCDATAcnf_CDATAreq AP_WRECOVERYind AP_WRECOVERYreq	always	never
AP_STREAM_FLAGS Bitmark indicating characteristics of the connection endpoint supporting the A/P-Library	unsigned long	None	bit values: AP_NDELAY	always	always

Name of attribute/ Purpose	**Data Type**	**Default value of attribute**	**Values legal for this attribute (* = not supported by current release)**	**Readable states (value of AP_STATE attribute)**	**Writable states (value of AP_STATE attribute)**

`AP_TOKENS_AVAIL`
Bitmark indicating which session tokens are available
	unsigned long	or'd bits: NULL	bit values:
			`AP_DATA_TOK *`
			`AP_SYNCMINOR_TOK`
			`AP_MAJACT_TOK *`
			`AP_RELEASE_TOK`
				always	never

`AP_TOKENS_OWNED`
Bitmark indicating which tokens the user currently controls
	unsigned long	N.A.	bit values:
			`AP_DATA_TOK *`
			`AP_SYNCMINOR_TOK`
			`AP_MAJACT_TOK *`
			`AP_RELEASE_TOK`
				in any states but:
				`AP_UNBOUND`
				`AP_IDLE`
				`AP_WASSOCcn_ ASSOCreq`
				`AP_WASSOCrsp _ASSOCind`
					never

Table A-2 AP_RCV Table of Attributes

Primitive	This primitive valid in states	Ap_env attribute that must be set	May change attributes	Next state (values of AP_STATE attribute)
A_ABORT_IND	all except: AP_UNBOUND AP_IDLE	none	AP_STATE	AP_IDLE
A_ASSOC_IND	AP_IDLE	AP_BIND_PADDR AP_LIB_SEL	AP_ACSE_SEL AP_CLD_AEID AP_CLD_AEQ AP_CLD_APID AP_CLD_APT AP_CLG_AEID AP_CLG_AEQ AP_CLG_APID AP_CLG_APT AP_CLG_CONN_ID AP_CNTX_NAME AP_INIT_SYNC_PT AP_LCL_PADDR AP_MODE_SEL AP_PCDL AP_PFU_SEL AP_PRES_SEL AP_REM_PADDR AP_ROLE_CURRENT AP_SESS_SEL AP_SFU_SEL AP_STATE AP_TOKENS_AVAIL AP_TOKENS_OWNED	AP_WASSOCrsp_ASSOCind

Primitive	This primitive valid in states	Ap_env attribute that must be set	May change attributes	Next state (values of AP_STATE attribute)
A_ASSOC_CNF	AP_WASSOCcnf_ASSOCreq	none	AP_ACSE_SEL AP_CLD_CONN_ID AP_CNTX_NAME AP_DCS AP_INIT_SYNC_PT AP_PFU_SEL AP_PCDRL AP_PRES_SEL AP_REM_PADDR AP_SESS_SEL AP_SFU_SEL AP_STATE AP_TOKENS_AVAIL AP_TOKENS_OWNED	(AP_IDLE, AP_DATAXFER)
A_PABORT_IND	all except: AP_UNBOUND AP_IDLE	none	AP_STATE	AP_IDLE
A_RELEASE_IND	AP_DATAXFER AP_WRELcnf_RELreq	none	AP_STATE	AP_WRELrsp_RELind (AP_WRELrsp_RELind_ init or AP_WRELcnf_RELreq_rsp)
A_RELEASE_CNF	AP_WRELcnf_RELreq AP_WRELcnf_RELreq_rsp	none	AP_STATE	(AP_IDLE or AP_DATAXFER) AP_WRELrsp_RELind
P_DATA_IND	AP_DATA XFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq	none	none	no state change
P_RESYNC_IND	AP_DATAXFER AP_WSYNCMAcnf_SYNCMAreq AP_WSYNCMArsp_SYNCMArsp AP_WACTEcnf_ACTEreq AP_WRECOVERYind AP_WRECOVERYreq AP_WRESYNCcnf_RESYNreq AP_WRELcnf_RELreq	none	AP_STATE	AP_WRESYNrsp_RESYNind

Primitive	This primitive valid in states	Ap_env attribute that must be set	May change attributes	Next state (values of AP_STATE attribute)
P_RESYNC_CNF	AP_WRESYNcnf_RESYNreq	none	AP_STATE AP_TOKENS_OWNED	AP_DATAXFER
P_SYNCMINOR_IND	AP_DATA_XFER	none	none	no state change
P_SYNCMINOR_CNF	AP_DATA_XFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTreq	none	none	no state change
P_TOKENGIVE_IND	AP_DATAXFER AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WSYNCHMArsp_ SYNCHMAind AP_WACTErsp_ACTEind AP_WRECOVERYind AP_WRECOVERYreq AP_WCDATAcnf_CDATAreq	none	AP_STATE AP_TOKENS_OWNED	no state change no state change no state change no state change no state change (no state change or AP_DATAXFER) (no state change or AP_DATAXFER) no state change
P_TOKENPLEASE_IND	AP_DATAXFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WCDATAcnf_CDATAreq	none	none	no state change
RO_ERROR_IND	AP_DATAXFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq	none	none	no state change
RO_INVOKE_IND	AP_DATAXFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq	none	none	no state change
RO_REJECTP_IND	AP_DATAXFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq	none	none	no state change

Primitive	This primitive valid in states	Ap_env attribute that must be set	May change attributes	Next state (values of AP_STATE attribute)

`RO_REJECTU_IND`	`AP_DATAXFER`
	`AP_WRELcnf_RELreq`
	`AP_WSYNCMAcnf_SYNCMAreq`
	`AP_WACTEcnf_ACTEreq`
		none	none	no state change

`RO_RESULT_IND`	`AP_DATAXFER`
	`AP_WRELcnf_RELreq`
	`AP_WSYNCMAcnf_SYNCMAreq`
	`AP_WACTEcnf_ACTEreq`
		none	none	no state change

Table A-3 AP_SND Table of Attributes

Primitive	This primitive, valid in states	Ap_env attributes that may be set (must be set in BOLD)	May change attributes	Next state (values of AP_STATE attribute)
A_ABORT_REQ	all except: AP_UNBOUND AP_IDLE	none	AP_STATE	AP_IDLE
A_ASSOC_REQ	AP_IDLE	**AP_BIND_PADDR** **AP_CNTX_NAME** **AP_LCL_PADDR** **AP_REM_PADDR** **AP_LIB_SEL** AP_ACSE_SEL AP_ROLE_ALLOWEDAP_ CLD_AEID AP_CLD_AEQ AP_CLD_APID AP_CLD_APT AP_CLG_AEID AP_CLG_AEQ AP_CLG_APID AP_CLG_APT AP_CLG_CONN_ID AP_MODE_SEL AP_PCDL AP_PFU_SEL AP_PRES_SEL AP_SESS_SEL AP_SFU_SEL AP_INIT_SYNC_PT	AP_ROLE_ CU RRENT AP_STATE AP_TOKENS_ OWNED	AP_WASSOCcnf_ ASSOCreq
A_ASSOC_RSP	AP_WASSOCrsp_ASSOCind	**AP_PCDRL** AP_CNTX_NAME AP_RSP_AEID AP_RSP_AEQ AP_RSP_APID AP_RSP_APT AP_CLD_CONN_ID AP_PFU_SEL AP_SESS_SEL AP_SFU_SEL AP_INIT_SYNC_PT	AP_DCS AP_STATE AP_TOKENS_ OWNED	(AP_IDLE, AP_DATAXFER)

Primitive	This primitive, valid in states	Ap_env attributes that may be set (must be set in BOLD)	May change attributes	Next state (values of AP_STATE attribute)
A_PABORT_REQ	all except: AP_UNBOUND AP_IDLE	none	AP_STATE	AP_IDLE
A_RELEASE_REQ	AP_DATAXFER	none	AP_STATE	AP_WRELcnf_RELreq
A_RELEASE_RSP	AP_WRELrsp_RELind AP_WRELrsp_RELind_init	none	AP_STATE	(AP_IDLE or AP_DATAXFER) AP_WRELcnf_RELreq
P_DATA_REQ	AP_DATAXFER AP_WRELrsp_RELind AP_WSYNCHMArsp_SYNCMAind AP_WACTErsp_ACTEind	none	none	no state change
P_RESYNC_REQ	AP_DATAXFER AP_WRELrsp_RELind AP_WRESYNrsp_RESYNind AP_WSYNCMAcnf_SYNCMAreq AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_WRECOVERYreq	none	AP_STATE	AP_IDLE
P_RESYNC_RSP	AP_WRESYNrsp_RESYNind	none	AP_STATE AP_TOKENS_ OWNED	AP_DATAXFER
P_SYNCMAJOR_REQ	AP_DATA_XFER	none	AP_SESS_VA AP_SESS_VM AP_SESS_VS C AP_STATE	AP_SYNCMAcnf_ SYN CMAreq
P_SYNCMAJOR_RSP	AP_SYNCrsp_SYNCMAind	none	AP_SESS_VA AP_SESS_VR AP_STATE	AP_DATA_XFER
P_SYNCMINOR_REQ	AP_DATA_XFER	none	none	no sate change
P_SYNCMINOR_RSP	AP_SYNCMArsp_SYNCMAind	none	none	no state change

Primitive	This primitive, valid in states	Ap_env attributes that may be set (must be set in BOLD)	May change attributes	Next state (values of AP_STATE attribute)
P_TOKENGIVE_REQ	AP_DATAXFER AP_WSYNCHMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_WRBCOVERYreq	none	AP_STATE AP_TOKENS_ OWNED	no state change no state change no state change no state change no state change (no state change or AP_DATAXFER)
P_TOKEN PLEASE_R EQ	AP_DATAXFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_WCDATArsp_CDATAind	none	none	no state change
RO_ERROR_REQ	AP_DATAXFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAindA P_WACTErsp_ACTEind	none	none	no state change
RO_INVOKE_REQ	AP_DATAXFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind	none	none	no state change
RO_RESULT_REQ	AP_DATAXFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind	none	none	no state change

Glossary

Application Layer Layer 7 of the OSI model; the user interface to network services and applications that provides services to directly support users.

ACSE Association Control Service Element. The service element in the OSI Application layer responsible for association establishment and release.

AE Title Application entity title. A unique name identifying an OSI service such as FTAM. The AE title consists of two parts, the AP-title and the AE- qualifier.

AFI Authority format identifier. The first part of the IDP definition of an NSAP. See IDP.

A/P ACSE/Presentation. Used to describe the ACSE/Presentation part of Hewlett-Packard's APRI implementation.

APLI ACSE/Presentation library interface (see APRI). Another term used to describe the ACSE/ Presentation part of Hewlett-Packard's APRI implementation.

APRI Abbreviation for ACSE/ Presentation and ROSE Interface. Hewlett-Packard's programmatic interface to the OSI services provided by ACSE, Presentation and remote operation service element (ROSE).

ARPA See DARPA.

ASN.1 Abstract Syntax Notation One. The OSI description language used to define data types. Abstract syntax describes data types independent of the underlying system used.

BCD binary coded decimal.

BSD Berkeley Software Distribution; a set of networking protocols developed primarily for user with the UNIX operating system. The Berkeley protocols are often used on the same network with ARPA.

CCITT See International Consultative Committee on Telephone and Telegraph

CLNS See Connectionless network service.

CMIP Common Management Information Protocol.

CMISE See Management Information Service Element.

Glossary

Common Management Information Protocol. the OSI network management protocol.

Common Element. The OSI application layer service element for network management.

conformance adherence to a product specification; along with interoperability, a defining characteristic of an OSI-compatible product.

Connectionless network service. The layer 3 network layer that provides datagrams to transmit data.

Connection oriented network service. The layer 3 network layer that provides end-to-end connection (virtual circuit) to transmit data.

CONS See Connection oriented network service.

Corporation for Open Systems

a non- profit, multinational consortium of computer users and vendors whose primary mission is to provide test procedures for conformance to OSI.

COS See Corporation for Open Systems

DARPA Defence Advanced Research Projects Agency; a branch of the Department of Defense (U.S.) that developed a set of networking protocols widely used in engineering and manufacturing.

Data Link Layer Layer 2 of the OSI model; establishes rules for transmission of data over the physical medium.

de facto standard a networking standard whose wide use makes it an unofficial industry standard.

draft international standard (DIS) the second stage of the process by which ISO develops standards; means the standard is complete and stable.

draft proposal (DP) the first stage in the development of ISO standards; means the developers believe the standard is technically stable.

ECMA European Computer Manufacturers Association.

Glossary

EDI Electronic Data Interchange; an emerging Application Layer standard for the electronic exchange of business data.

ES End system.

EWOS European Workshop on Open Systems.

FDDI Fiber Distributed Data Interface; a standard for a local area network that uses fiber optics and operates at 100 Mbps.

FTAM File Transfer, Access and Management; an Application Layer standard that enables users to transfer and access files within a multivendor network.

GOSIP Government OSI Profile; a set of OSI specifications that the U.S. Government is using for its network procurement.

IEEE 802 a series of standards for local area networking developed by the Institute of Electrical and Electronics Engineers (IEEE).

implementors agreement

accord reached by computer users and vendors on how they will implement a networking standard so that their products can communicate. Also known as functional standard or standard profiles.

international standard (IS) a networking standard that has been approved by ISO or CCITT.

International Consultative Committee on Telephone and Telegraph division of the U.N. International Telecommunications Union that coordinates standard-setting activities.

interoperability the ability of different vendors' products to communicate along a network; along with conformance, a requirement of an OSI product.

Integrated Services Digital Network a developing standard whose goal is to permit the integration of data networks with telephone communications and other services (e.g., video) by means of digital technology.

IDI Initial domain identifier. The second part of the initial domain part (IDP) definition of an NSAP. See IDP.

IDP Initial domain part. The IDP part of an NSAP identifies which national or international group has defined the NSAP format for a given NSAP. It is partitioned into two parts, the authority format identifier (AFI) and the initial domain identifier (IDI). IDP spaces are managed by organizations such as AFNOR, ANSI and NIST.

ISDN See Integrated Services Digital Network

IS-IS Protocol Intermediate system to intermediate system protocol. An evolving ISO protocol standard to provide automatic routing between intermediate systems.

ISO International Standard Organization; the international body that is coordinating the effort to establish OSI standards for multivendor networking.

local area network (LAN) a data communications network of limited size- within a building, group of buildings or campus.

MAP Manufacturing Automation Protocol; a subset of OSI standards that specifies networking protocols for manufacturing.

migration the non-disruptive replacement of proprietary networks with standards-based networks.

MMS Manufacturing Messaging Service; a networking standard that defines methods for the provision of specific applications for manufacturing.

multivendor networking the linking of hardware, software, computers and peripherals made by different vendors into a single communications network.

NIST National Institute of Standards and Technology (formerly NBS). Runs implementors' workshops that develop agreements among vendors on which subsets of standards will be implemented.

network a combination of hardware and software that allows the transfer of data, both locally and over great distances, between two or more systems.

Network Layer Layer 3 of the OSI model; governs the routing and switching of data among networks.

Glossary

ODA Office Document Architecture; a developing standard that enables text, graphics and facsimile, in different document formats, to be moved over a multivendor network.

OSI Open Systems Interconnection; the name used to describe a set of data communications standards which have been agreed upon at an international level by national standards bodies.

OSI Reference Model a modular, seven-layer construct that specifies how data will travel among systems of a multivendor network, as well as within a single system.

Physical Layer Layer 1 of the OSI model; enables the sending and receiving of bits between nodes on a network.

Presentation Address An address made up of a P-selector, S-selector and T-selector (presentation, session, and transport selectors) and one or more NSAPs. Optionally, it can include the application entity (AE) title which is a unique name identifying an OSI service such as FTAM.

Presentation Layer Layer 6 of the OSI model; where data are put into usable form.

proprietary networking

networking that uses the products and protocols of a single vendor.

protocol a formalized set of rules by which computers communicate.

ROSE Remote Operations Service Element. The OSI application service element which manages request/reply interactions.

SAP Service Access Point. A SAP is a pipe between two OSI layers such that an entity at layer n + 1 can obtain a set of services from layer n.

Selector a sequence of octets (bytes) used to identify a SAP.

Session Layer Layer 5 of the OSI model; permits the setup of a communications path along a network and manages the coordination between processes.

SNA Systems Network Architecture; IBM's proprietary networking formats, protocols and procedures that are also a de facto networking standard.

Glossary

SPAG Standards Promotion and Applications Group; based in Europe and working with COS on the development and promotion of conformance tests procedures for OSI products.

TCP transmission control protocol. A network protocol that establishes and maintains connections between nodes. TCP is the transport protocol used in ARPA networks. Internet protocol (IP) addresses are used to identify systems in an ARPA network.

Transport Layer Layer 4 of the OSI model; provides end-to-end data integrity between processes.

United Kingdom GOSIP The United Kingdom defined set of OSI standards which includes a specific NSAP format.

wide area network a network that covers a large area; can be as large as the entire world.

X.25 an international networking standard, developed by the CCITT, for the connection of computer equipment to packet switching networks.

X.400 CCITT standard for electronic message handling systems (e.g., electronic mail) in multivendor environments.

X.500 Directory Services An international OSI standard for distributed directory services for multivendor OSI environments. Directory Services refer to any service which stores information about people or things in the world. The most common example of a directory service is the phone book produced by the telephone company.

Index

Index

Index