

*SunLink[®] OSI 8.1
API Programmer's Reference*



SunSoft

A Sun Microsystems, Inc. Business

2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No.: 802-2040-11
Revision A, March 1995

© 1995 Sun Microsystems, Inc.
2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

All rights reserved. This product and related documentation are protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Portions of this product may be derived from the UNIX[®] and Berkeley 4.3 BSD systems, licensed from UNIX System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc., and the University of California, respectively. Third-party font software in this product is protected by copyright and licensed from Sun's font suppliers.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7013 (c)(1)(ii) and FAR 52.227-19.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

TRADEMARKS

Sun, the Sun logo, Sun Microsystems, Solaris and SunLink are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and certain other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. OPEN LOOK is a registered trademark of Novell, Inc. PostScript and Display PostScript are trademarks of Adobe Systems, Inc. All other product names mentioned herein are the trademarks of their respective owners.

All SPARC trademarks, including the SCD Compliant Logo, are trademarks or registered trademarks of SPARC International, Inc. SPARCstation, SPARCserver, SPARCengine, SPARCstorage, SPARCware, SPARCcenter, SPARCclassic, SPARCcluster, SPARCdesign, SPARC811, SPARCprinter, UltraSPARC, microSPARC, SPARCworks, and SPARCcompiler are licensed exclusively to Sun Microsystems, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK[®] and Sun[™] Graphical User Interfaces were developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

X Window System is a product of the Massachusetts Institute of Technology.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.



Contents

Preface.....	xv
1. Overview	1
OSI Upper Layer Services.....	1
OSI Reference Model.....	2
Application Layer Services.....	3
Presentation Layer Services	3
Session Layer Services.....	4
OSI Service Primitives.....	5
APLI Basic Model.....	7
APLI Service User	8
APLI Service Provider.....	8
APLI Environment.....	8
APLI Instance.....	9
APLI Functions	9
APLI Service Primitives	10

2. Developing Applications with APLI	11
Basic Program Development Summary	12
Opening an APLI Instance	15
Initializing an APLI Instance	15
Configuring the APLI Environment	15
Setting the Presentation Addresses	16
Negotiating the Presentation Contexts	16
Sending and Receiving APLI Service Primitives	17
Sending Protocol Information	18
Sending User Data	19
Setting the Execution Mode	20
Closing an APLI Instance	20
Setting the Token Assignment	21
Housekeeping and Auxiliary Functions	22
Releasing Allocated Memory	22
Polling the APLI Instance	22
Reporting Errors	22
Error Summary	23
Compiling and Linking APLI Applications	25
Runtime Libraries	25
External Include Files	25
Compile and Link Procedure	26
3. APLI Functions	27
APLI Function Summary	28

ap_open()—open APLI instance	29
ap_init_env()—initialize APLI environment attributes . . .	31
ap_get_env()—get APLI environment attribute	33
ap_set_env()—set APLI environment attribute	35
ap_snd()—send primitive	37
ap_rcv()—receive primitive	40
ap_free()—free allocated memory	43
ap_poll()—poll APLI events	45
ap_error()—return error message.	47
ap_close()—close APLI instance.	49
4. APLI Environment Attributes.	51
APLI Attribute Summary	52
AP_ACSE_AVAIL	54
AP_ACSE_SEL	54
AP_BIND_PADDR	55
AP_CLD_AEID	55
AP_CLD_AEQ.	55
AP_CLD_APID	56
AP_CLD_APT.	56
AP_CLD_CONN_ID.	56
AP_CLG_AEID	57
AP_CLG_AEQ.	57
AP_CLG_APID	57
AP_CLG_APT.	58

AP_CLG_CONN_ID	58
AP_CNTX_NAME	58
AP_DCS	59
AP_DPCN	59
AP_DPCR	60
AP_INIT_SYNC_PT	60
AP_LCL_PADDR	60
AP_LIB_AVAIL	61
AP_LIB_SEL	61
AP_MODE_AVAIL	61
AP_MODE_SEL	62
AP_MSTATE	62
AP_PCDL	62
AP_PCDRL	63
AP_PFU_AVAIL	64
AP_PFU_SEL	64
AP_PRES_AVAIL	64
AP_PRES_SEL	65
AP_REM_PADDR	65
AP_ROLE_ALLOWED	65
AP_ROLE_CURRENT	66
AP_RSP_AEID	67
AP_RSP_AEQ	67
AP_RSP_APIID	67

AP_RSP_APT.....	68
AP_SESS_AVAIL.....	68
AP_SESS_SEL.....	68
AP_SFU_AVAIL.....	69
AP_SFU_SEL.....	69
AP_STATE.....	70
AP_STREAM_FLAGS.....	70
AP_TOKENS_AVAIL.....	71
AP_TOKENS_OWNED.....	71
Environment Data Types.....	72
ap_cdl_t.....	72
ap_cdr1_t.....	73
ap_conn_id_t.....	75
ap_dcn_t.....	76
ap_dcs_t.....	77
ap_paddr_t.....	78
any_t.....	79
objid_t.....	80
Environment Read/Write Table.....	81
5. APLI Service Primitives.....	89
A_ABORT_REQ—request abnormal release.....	93
A_ABORT_IND—indicate abnormal release.....	95
A_ASSOC_REQ—request association.....	97
A_ASSOC_IND—indicate association request.....	100

A_ASSOC_RSP—respond to association request	102
A_ASSOC_CNF—confirm association request	106
A_PABORT_REQ—request abnormal provider release	110
A_PABORT_IND—indicate abnormal provider release	114
A_RELEASE_REQ—request normal release	118
A_RELEASE_IND—indicate normal release request	120
A_RELEASE_RSP—respond to normal release request	122
A_RELEASE_CNF—confirm normal release request	124
P_DATA_REQ—send user data	126
P_DATA_IND—indicate receipt of user data	128
P_TOKENGIVE_REQ—give session token	130
P_TOKENGIVE_IND—indicate receipt of token	132
P_TOKENPLEASE_REQ—request token	134
P_TOKENPLEASE_IND—indicate request for token	136
A. APLI Example Application	139
Example Files	140
Modifying the Local Environment File (enviro.h)	141
Compiling and Linking the Example Code	143
Running the Example Application	143
B. Referenced Documents	145
Glossary	149
Index	153

Figures

Figure 1-1	OSI Upper and Lower Layer Infrastructure	2
Figure 1-2	Confirmed Service	6
Figure 1-3	Unconfirmed Service.	6
Figure 1-4	APLI Basic Model	7
Figure 2-1	Custom Applications using APLI	12
Figure 2-2	Basic Program Development	14
Figure 2-3	Establishing an Association	17

Tables

Table 2-1	API Service Primitives	17
Table 2-2	Tokens Set by Requestor.	21
Table 2-3	Tokens Set by Acceptor	21
Table 2-4	API Error Codes	23
Table 3-1	API Function Summary	28
Table 4-1	Attribute Summary	52
Table 4-2	Attribute Read/Write Table.	81
Table 5-1	Primitive/State/Attribute Relationships.	90

Code Samples

Code Fragment 3-1	Using <code>ap_open()</code>	30
Code Fragment 3-2	Using <code>ap_init_env()</code>	32
Code Fragment 3-3	Using <code>ap_get_env()</code>	34
Code Fragment 3-4	Using <code>ap_set_env()</code>	36
Code Fragment 3-5	Using <code>ap_snd()</code>	39
Code Fragment 3-6	Using <code>ap_rcv()</code>	42
Code Fragment 3-7	Using <code>ap_free()</code>	44
Code Fragment 3-8	Using <code>ap_poll()</code>	46
Code Fragment 3-9	Using <code>ap_error()</code>	48
Code Fragment 3-10	Using <code>ap_close()</code>	50

Preface

The ACSE/Presentation Programming Interface (APLI) provides a standard interface between an application entity (as defined by the OSI reference model) and the services provided by ACSE and the presentation layer of the OSI protocol stack.

The *SunLink OSI 8.1 APLI Programmer's Reference* provides a comprehensive description of the SunLink implementation of this programming interface, which is used specifically to access the services provided by the ACSE and presentation layer components of the SunLink OSI Communication Platform. It provides an overview of the programming model and a detailed description of each of the functions, service primitives, and data structures supported by this implementation of APLI.

Who Should Use This Book

This book is intended for application developers and system administrators who are already familiar with the OSI reference model and the services provided by the layers in the protocol stack.

A more detailed description of the OSI reference model and instructions on how to configure and use the SunLink OSI Communication Platform is contained in the *SunLink OSI 8.1 Communication Platform Administrator's Guide*.

How This Book Is Organized

Chapter 1, “Overview,” introduces the SunLink implementation of APLI, which is used to develop custom applications that run over ACSE and the Presentation layer of the SunLink OSI Communication Platform (stack).

Chapter 2, “Developing Applications with APLI,” describes the mechanisms that APLI uses to communicate with the ACSE and presentation layer services and describes how to develop applications that access these services.

Chapter 3, “APLI Functions,” defines the functions provided by the SunLink implementation of APLI. The function definitions are presented in the form of *manual pages* that provide detailed specifications of parameters and structures.

Chapter 4, “APLI Environment Attributes,” describes the environment attributes that are supported by the SunLink implementation of APLI, and includes a description of the associated data structures.

Chapter 5, “APLI Service Primitives,” describes the service primitives supported by the SunLink implementation of APLI. Each primitive corresponds to one of the services provided by the underlying OSI protocol.

Appendix A, “APLI Example Application,” describes the APLI example files provided with the SunLink OSI Communication Platform and explains how to compile and run the APLI example application.

Appendix B, “Referenced Documents,” lists the various OSI documents and specifications referenced in this manual.

Glossary is a list of words and phrases found in this book and their definitions.

Related Books

The other books in the SunLink OSI documentation set are:

SunLink OSI 8.1 Communication Platform Administrator’s Guide
(Part No. 801-4975-13)

SunLink OSI 8.1 TLI Programmer’s Reference
(Part No. 801-7170-11)

See also *Installing and Licensing SunLink OSI 8.1* for detailed instructions on how to load the product software from the CD-ROM.

What Typographic Changes and Symbols Mean

The following table describes the type changes and symbols used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. system% You have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	<div>system% su Password::</div>
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.
Code samples are included in boxes and may display the following:		
%	UNIX C shell prompt	system%
\$	UNIX Bourne and Korn shell prompt	system\$
#	Superuser prompt, all shells	system#

Overview



<i>OSI Upper Layer Services</i>	<i>page 1</i>
<i>APLI Basic Model</i>	<i>page 7</i>

The ACSE/Presentation Programming Interface (APLI) provides a standard interface between a custom application and the services provided by ACSE and the presentation layer of the OSI protocol stack.

This chapter introduces the SunLink implementation of APLI, which is used to develop custom applications that run over ACSE and the presentation layer of the SunLink OSI Communication Platform. It provides a brief review of the services provided by the upper layers of the OSI protocol stack, and describes the basic model on which APLI is based.

OSI Upper Layer Services

To fully understand the subsequent chapters of this manual, you must be familiar with the services provided by the upper layers of the underlying OSI protocol stack. This section provides a brief overview of these services and includes references to the OSI specifications for the services and protocols under discussion.

A more detailed description of the OSI reference model and instructions on how to configure and use the SunLink OSI Communication Platform is contained in the *SunLink OSI 8.1 Communication Platform Administrator's Guide*.

OSI Reference Model

Figure 1-1 provides a simplified view of the layers in the OSI reference model (ISO 7498). Within this model, the transport layer and the layers below cooperate to provide an end-to-end transmission path using available networks. These layers are network-dependent; functions such as error detection and correction, flow control, and sequencing may occur at different layers, depending on the types of network involved. (For example, these functions may be provided by a connection-oriented data link layer, an X.25-based network layer, or by the transport layer.)

The layers above the transport layer can be regarded as network-independent since the transport layer provides a consistent service regardless of the variations in the layers below. These layers are referred to as the Upper Layers and consist of the session layer, the presentation layer, and the application layer (which is organized internally as a set of applications and *application-service-elements* including ACSE). The services provided by the layers in the upper layer infrastructure are discussed individually in the following sections.

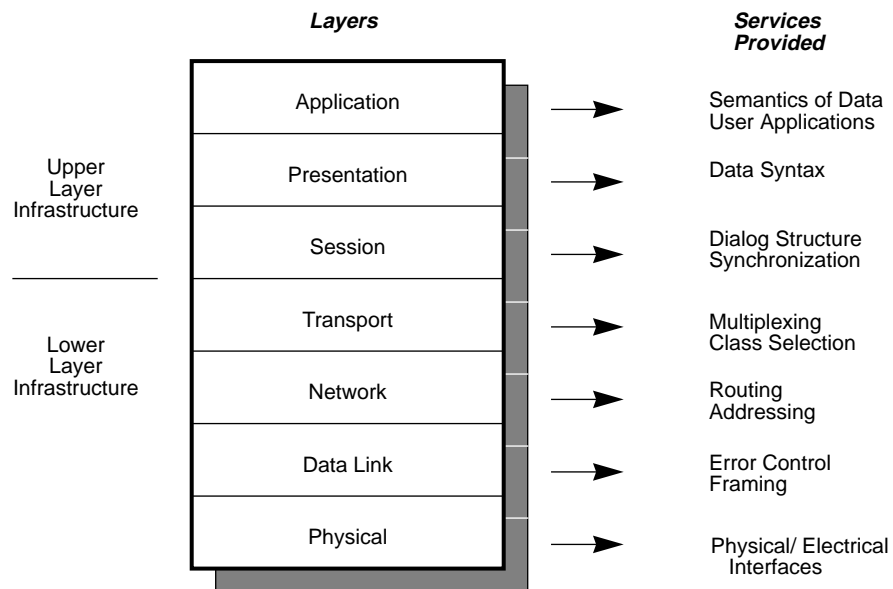


Figure 1-1 OSI Upper and Lower Layer Infrastructure

Application Layer Services

The application layer contains the OSI applications that access the services of an OSI protocol stack. The OSI reference model describes communications between *application-entities* (defined as the aspects of an application process pertinent to OSI). Each application-entity consists of the communications aspects of the application combined with a number of *application-service-elements*, which are part of the application layer and provide OSI services to the user. An application-service-element may be specific to a particular application (for example, the Common Management Information Service Element (CMISE), which provides information and command exchange services for use by management applications), or generally useful to a range of applications (for example, ACSE, which provides association control services that establish and release associations between application entities). Each service element uses the services of the presentation layer and possibly other service elements to implement its protocol and provide its service. For example, a File Transfer Access and Management (FTAM) initiator relies upon ACSE to establish an association with a remote FTAM responder. Presentation layer primitives are then used to pass FTAM Protocol Data Units (PDUs) to the peer application-entity.

The ACSE *application-service-element* (**ISO 8649** and **ISO 8650**) defines service primitives that map directly to the equivalent primitives of the presentation layer with a number of additional parameters, used to identify the application-entities involved in an association and to agree the application context in which communication is to occur (that is, the set of application-service-elements and associated information required for a particular application).

Presentation Layer Services

The presentation layer (**ISO 8822** and **ISO 8823**) is responsible for the representation of data in transit between application entities. The main purpose of this representation is to preserve the meaning of the data transferred over an association. This allows cooperating application entities to exchange data without being concerned about differences of representation of data objects (for example, byte-ordering and integer-size).

The service primitives available at this layer map directly to those defined for the session layer; the presentation layer transforms the data units passed to it by an application entity from the local syntax into an agreed *transfer syntax* and

then passes the resulting data units to the session layer. The peer presentation layer performs the opposite transformation before passing the resulting data unit to its application entity.

The presentation layer negotiates, on behalf of the application entity, a set of *abstract syntaxes* and *transfer syntaxes* to use during the association.

An abstract syntax is defined by a protocol specification to describe the structure of the Protocol Data Units that are transferred by the protocol. Abstract syntaxes are expressed using ASN.1, which is specified in ISO/IEC 8824. For example, an FTAM application entity requesting a presentation connection would specify the abstract syntaxes for ACSE (to represent the ACSE data units), FTAM (to represent the FTAM data units), and one or more syntaxes for the FTAM document types to use during this FTAM session.

Each data value passed to the presentation layer specifies the presentation context to which it belongs (the abstract syntax name). This provides the information required by the presentation layer to encode the data value into the transfer syntax. A data value may contain embedded values that belong to other syntaxes. Again, the abstract syntax name for the embedded value is sufficient to allow the presentation layer to encode it.

Session Layer Services

The session layer (**ISO 8326** and **ISO 8327**) enhances the basic end-to-end service of the transport layer by providing services that allow applications to organize and synchronize their interactions and data transfers. These services are supported by a set of tokens and service primitives, the use of which is negotiated during session connection.

In addition to connection establishment and data transfer services that map closely to those of the transport layer, the session layer provides the following services:

- **Orderly Release Service.**

This service is used to ensure that session release occurs without loss of data. To request/refuse a session release, an application must control the *release* token.

- **Synchronization Services.**

These services allow data transfer to be controlled for recovery purposes. The major synchronization point service divides a data exchange into dialog units; the data in each dialog unit is confirmed as received correctly before the next unit can be started. The major and minor synchronization point services identify points in the data transfer at which recovery may occur. The resynchronization service allows the data transfer to be restarted at an identified synchronization point (not earlier than the last confirmed synchronization point). The right to issue synchronization point requests is determined by control of the *synchronize-minor* token and *major/activity* token respectively.

- **Activity Services.**

These services allow a data exchange to be divided into distinct logical pieces of work, bracketed by *activity start* and *activity end* requests. Activities can be divided into dialog units by using the major synchronize service, and can be interrupted and resumed. Again, the right to start or end an activity is determined by control of the *major/activity* token.

- **Half-Duplex Data Transfer Service.**

This service allows session service users to take turns at transferring data over a session connection. Permission to transfer data is controlled by possession of the *data* token.

OSI Service Primitives

Sequences of events that occur between adjacent layers are defined by a set of named *service primitives*. These service primitives are classified into four types that define the handshaking mechanism:

Service Request

Issued by the initiator of an action to request a service from the responder.

Service Indication

Received by the responder to an action to indicate a request for service.

Service Response

Issued by the responder to an action (following the receipt of a service indication) to accept or reject the request for service.

Service Confirmation

Received by the initiator of an action to confirm the acceptance or rejection of a request for service.

For a *confirmed negotiation*, all four service primitives are issued as shown in Figure 1-2.

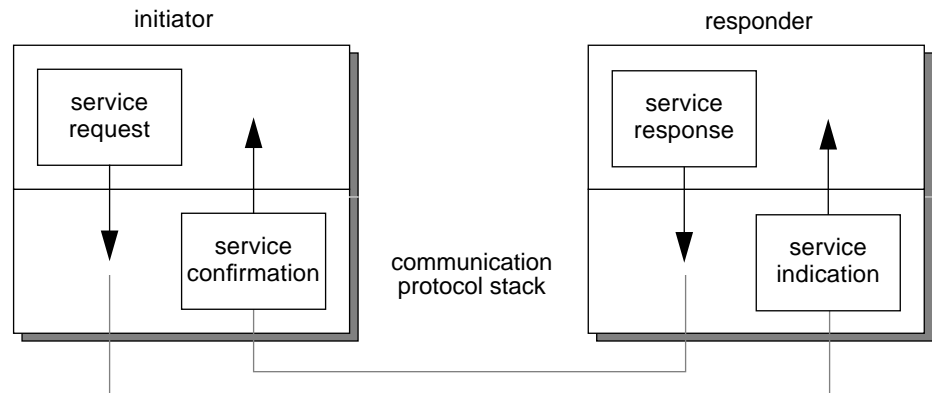


Figure 1-2 Confirmed Service

For an *unconfirmed negotiation*, only the first part of the handshaking mechanism is used as shown in Figure 1-3.

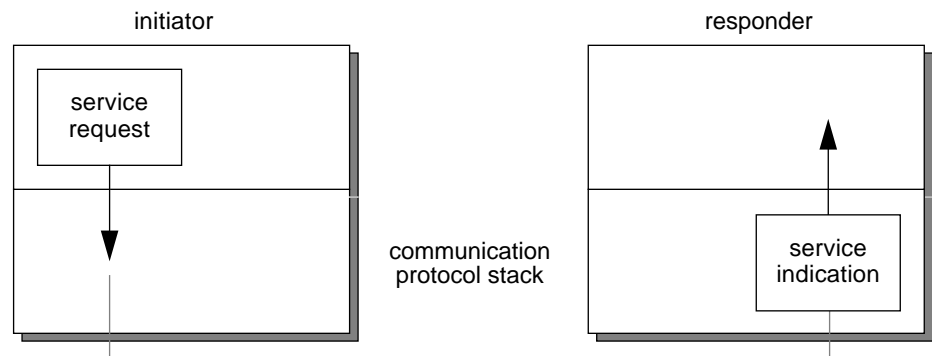


Figure 1-3 Unconfirmed Service

APLI Basic Model

Figure 1-4 shows how the APLI model provides a standard interface between an application-entity (the *APLI service user*) and the services provided by ACSE and the presentation layer (the *APLI service provider*).

APLI provides *functions* that are used to establish associations and transfer data based on ACSE and presentation layer services, and an *environment* that is used to store information that concerns the current association. Requests and responses from the service user are combined with information from the APLI environment and passed as ACSE or presentation service primitives to the service provider. Indications and confirmations from the service provider update the APLI environment and are passed to the service user.

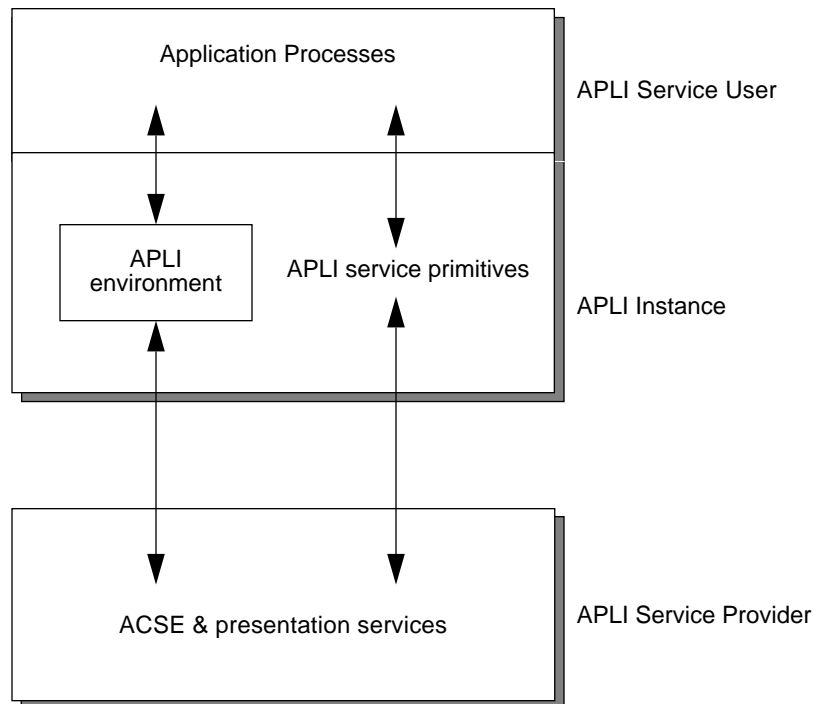


Figure 1-4 APLI Basic Model

APLI Service User

The APLI service user is an application that uses ACSE and presentation layer services to implement its functions. This service user corresponds to the concept of an application-entity, as defined by the OSI reference model (ISO 7498), and as such must have a unique presentation address. Depending on the requirements of the application, the service user may be an independent program, or a group of cooperating processes.

APLI Service Provider

The APLI *service provider* is an implementation of the ACSE and presentation layers. In the case of the SunLink implementation of APLI, it is assumed that the elements of the service provider are supplied by the device `/dev/ooapi`, which corresponds to the SunLink OSI Communication Platform.

APLI Environment

The APLI environment stores all of the information necessary to establish and maintain an association with another application entity in the form of *attributes*. APLI environment attributes are divided into two types:

- *Read-only* attributes store information that reflect the current status of the association, OSI protocol stack, or APLI instance.
- *Read-write* attributes can be modified by the service user and are used to specify the characteristics of an association or the APLI instance.

Certain attributes are only accessible (for either reading or writing) when the APLI instance is in a given state. For example, the environment attribute `AP_REM_PADDR`, which is used to set the presentation address for a remote application entity, can only be modified when the APLI instance is idle (`AP_IDLE`) or unbound (`AP_UNBOUND`). The current status of the APLI instance is returned in the attribute `AP_STATE`.

See Chapter 4, “APLI Environment Attributes” for a comprehensive description of each of the APLI environment attributes supported by the SunLink implementation of APLI.

APLI Instance

An APLI *instance* is the collection of information and OSI communications capabilities required to establish and maintain a single application association. The instance includes the APLI *environment* and this must be initialized before the association can be used to communicate. The service user may have several APLI instances in progress at one time; however, each APLI instance supports only one association.

When the APLI instance is opened, an APLI *instance identifier* is returned which is used to identify this instance in subsequent calls to APLI functions. This identifier is meaningful only within the context of the process that created the *instance*. In the SunLink implementation of APLI, the instance identifier is passed as an **integer** file descriptor (*fd*) that points to the device `/dev/oopi`, which corresponds to the SunLink OSI Communication Platform.

APLI Functions

APLI *functions* are used to establish and initiate an APLI instance, to manage the APLI environment and resources, and to transfer APLI service primitives. The functions supported by the SunLink implementation of APLI can be divided into the following categories:

- Functions used to establish and release an APLI instance:
`ap_open()`, `ap_close()`
- Functions used to manage the APLI environment:
`ap_init_env()`, `ap_set_env()`, `ap_get_env()`
- Functions used to send/receive APLI service primitives:
`ap_snd()`, `ap_rcv()`
- Function used to manage memory allocated to the APLI environment:
`ap_free()`
- Function used to retrieve status information:
`ap_poll()`
- Function used to retrieve error messages corresponding to specific error return codes:
`ap_error()`

See Chapter 3, “APLI Functions” for a comprehensive description of each of these functions and how to use them to develop a custom application.

APLI Service Primitives

Each APLI service primitive corresponds to an ACSE or presentation layer service. Primitives that correspond to an ACSE service are prefixed with A_ (for example, A_ASSOC_REQ) and primitives that correspond to a presentation layer service are prefixed with P_ (for example, P_DATA_REQ).

With the exception of *user data* (discussed in the following chapter), the parameters passed to the service provider by APLI when sending a particular primitive are derived from attributes held in the APLI environment and from *protocol information* passed to APLI by the service user. Protocol information allows the service user to pass parameters specific to each primitive and to override some of the values held in the environment.

When receiving primitives from the service provider, APLI updates the values of some environment attributes and passes specific parameters to the service user along with the primitive itself.

The service primitives supported by the SunLink implementation of APLI can be divided into the following categories:

- Primitives used to establish an association (A-ASSOCIATE):
A_ASSOC_REQ, A_ASSOC_IND, A_ASSOC_RSP, A_ASSOC_CNF
- Primitives used to release an association (A-RELEASE):
A_RELEASE_REQ, A_RELEASE_IND, A_RELEASE_RSP, A_RELEASE_CNF
- Primitives used to abort an association (A-ABORT):
A_ABORT_REQ, A_ABORT_IND
- Primitives used to abort the presentation provider (A-P-ABORT):
A_PABORT_REQ, A_PABORT_IND
- Primitives used to send user-data (P-DATA):
P_DATA_REQ, P_DATA_IND
- Primitives used to give up tokens (P-TOKEN-GIVE):
P_TOKENGIVE_REQ, P_TOKENGIVE_IND
- Primitives used to request tokens (P-TOKEN-PLEASE):
P_TOKENPLEASE_REQ, P_TOKENPLEASE_IND

See Chapter 5, “APLI Service Primitives” for a comprehensive description of each of the APLI service primitives and how to use them to access ACSE and presentation layer services.

Developing Applications with APLI

2 

<i>Basic Program Development Summary</i>	<i>page 12</i>
<i>Opening an APLI Instance</i>	<i>page 15</i>
<i>Initializing an APLI Instance</i>	<i>page 15</i>
<i>Configuring the APLI Environment</i>	<i>page 15</i>
<i>Sending and Receiving APLI Service Primitives</i>	<i>page 17</i>
<i>Closing an APLI Instance</i>	<i>page 20</i>
<i>Setting the Token Assignment</i>	<i>page 21</i>
<i>Housekeeping and Auxiliary Functions</i>	<i>page 22</i>
<i>Compiling and Linking APLI Applications</i>	<i>page 25</i>

This chapter describes the mechanisms that APLI uses to communicate with the ACSE and presentation layer services and describes how to develop applications that access these services.

The SunLink implementation of APLI provides a standard interface between custom applications and the ACSE and presentation layers of the SunLink OSI Communication Platform. Figure 2-1 shows how applications developed using APLI access the ACSE and presentation services through this interface. Once an APLI instance is established and initialized, APLI service primitives are passed to and from the ACSE and presentation layers using the APLI functions `ap_snd()` and `ap_rvc()`.

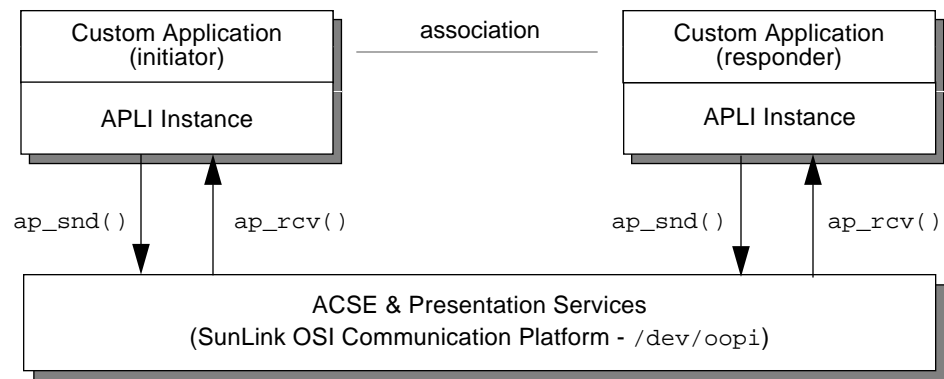


Figure 2-1 Custom Applications using APLI

Basic Program Development Summary

Figure 2-2 on page 14 illustrates the basic steps required to establish an association with a remote application entity using APLI. The procedure presented is intended solely as a general description of how the interface might be used. It should not be construed as an attempt to provide a template for constructing any particular application. Moreover, it is assumed that the service user is familiar with the ACSE and presentation layer protocols and understands the role of the ACSE service user in establishing, using, and terminating an association between two application entities.

1. **Open an APLI instance using `ap_open()`.**
2. **Initialize the APLI environment using `ap_init_env()`.**
3. **Modify the default APLI environment to bind a presentation address and configure the APLI instance using `ap_get_env()` and `ap_set_env()`.**
4. **Use `ap_snd()` and `ap_rcv()` to exchange service primitives with the ACSE and presentation layers (for example, establish an association, transfer data, release an association).**
5. **Modify the default APLI environment to reconfigure the APLI instance using `ap_set_env()`.**

-
6. Use `ap_snd()` and `ap_rcv()` to exchange service primitives with the ACSE and presentation layers (for example, establish an association, transfer data, release an association).
 7. Close the APLI instance using `ap_close()`.

In addition to the basic functions listed in the summary of steps, an application entity must also perform basic housekeeping using auxiliary APLI functions to manage the memory assigned to the APLI environment and to return error messages as required.

This series of events is illustrated by the example application located in the directory `/opt/SUNWconn/osinet/example`.

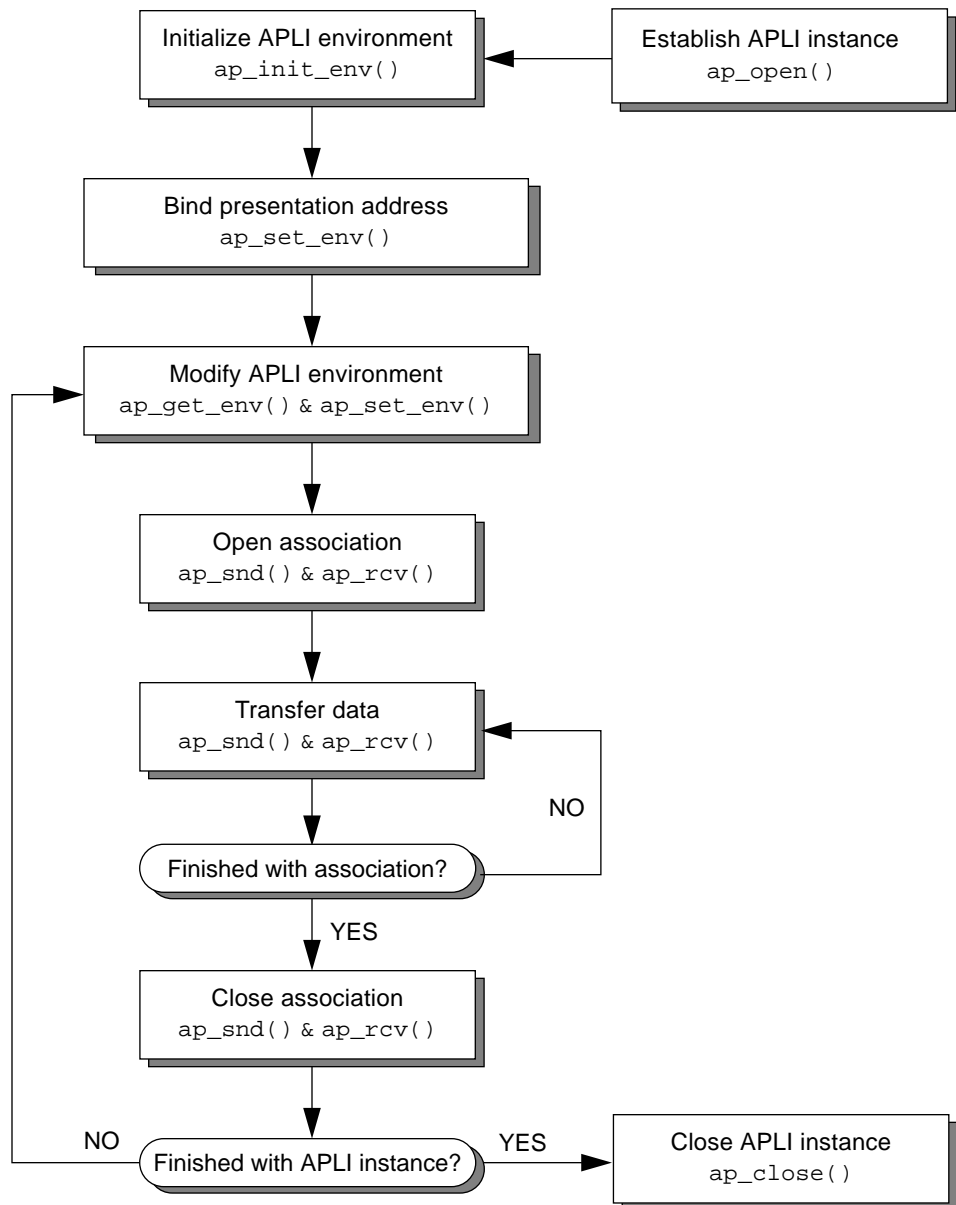


Figure 2-2 Basic Program Development

Opening an APLI Instance

To access the service provider, the service user must first open an APLI instance using the `ap_open()` function. The parameters passed to `ap_open()` identify the service provider and set some of the default behavior for the APLI instance. In the case of the SunLink implementation of APLI, the service provider must be the device `/dev/oopi` that corresponds to the SunLink OSI Communication Platform. Note that a single service user can establish multiple, concurrent APLI instances.

The function `ap_open()` returns an **integer** file descriptor (*fd*) that is used to identify the APLI instance in all future calls to APLI functions.

See also “`ap_open()`—open APLI instance” on page 29.

Initializing an APLI Instance

After an APLI instance is opened, the service user must initialize the APLI environment by calling the function `ap_init_env()`. This function assigns the resources required by the APLI environment and resets all of the attributes to their default values, as listed in Chapter 4, “APLI Environment Attributes.”

See also “`ap_init_env()`—initialize APLI environment attributes” on page 31.

Configuring the APLI Environment

After the APLI environment has been initialized, it must be modified to identify and describe the specific association to be opened. Each APLI instance can support exactly one association at a time; however, once an association is terminated, the service user can modify the environment and open another association without terminating the APLI instance.

The APLI environment is configured using calls to `ap_get_env()` to return the current state of the environment, and calls to `ap_set_env()` to modify the attributes. The configuration of the APLI environment is entirely application-dependent and there are a certain number of environment attributes that must be set by every application (for example, the presentation addresses).

See also “`ap_get_env()`—get APLI environment attribute” on page 33 and “`ap_set_env()`—set APLI environment attribute” on page 35.

Setting the Presentation Addresses

As specified in the OSI reference model (**ISO 7498**), an application entity must be addressable via a single, globally unique, presentation address. Thus, each APLI service user must have a unique presentation address; however, multiple APLI instances created by the same APLI service user may share the same presentation address.

A presentation address is bound to the APLI instance by setting the `AP_BIND_PADDR` attribute. This attribute must be set before the APLI user can send or receive any service primitives.

If the APLI user is an association-initiator, it must also specify the presentation address of the association-responder, by setting the `AP_REM_PADDR` attribute, before issuing an association request.

Negotiating the Presentation Contexts

An important aspect of association establishment is the negotiation of the presentation contexts to be used during the association. It is the responsibility of the service user to define the presentation contexts used to exchange user data.

Before requesting an association, the service user may set the Presentation Context Definition List (`AP_PCDL`) and the Default Presentation Context Name (`AP_DPCN`) to indicate the presentation contexts that it will propose to the association-responder. Since the service user is responsible for encoding and decoding all user data, the Presentation Context Definition List (`AP_PCDL`) must include the proposed transfer syntaxes for each proposed abstract syntax. APLI ensures that the presentation context used to exchange ACSE protocol information between peer entities is available. The presentation context can also be set by the service user.

Similarly, before responding to an association request, the service user may set the Presentation Context Definition Response List (`AP_PCDRL`) and the Default Presentation Context Result (`AP_DPCR`) to indicate which of the presentation contexts proposed by the association-initiator it will accept or reject.

Each entry in the Presentation Context Definition List (`AP_PCDL`) has a corresponding entry in the Presentation Context Definition Response List (`AP_PCDRL`) that indicates whether it is accepted or rejected.

Sending and Receiving APLI Service Primitives

Each APLI service primitive corresponds to an ACSE or presentation layer service as shown in Table 2-1. Primitives that correspond to an ACSE service are prefixed with **A_** and primitives that correspond to a presentation layer service are prefixed with **P_**.

Table 2-1 APLI Service Primitives

ACSE/Presentation Services	APLI Service Primitives
A-ASSOCIATE	A_ASSOC_REQ/A_ASSOC_IND A_ASSOC_RSP/A_ASSOC_CNF
A-RELEASE	A_RELEASE_REQ/A_RELEASE_IND A_RELEASE_RSP/A_RELEASE_CNF
A-ABORT	A_ABORT_REQ/A_ABORT_IND
A-P-ABORT	A_PABORT_REQ/A_PABORT_IND
P-DATA	P_DATA_REQ/P_DATA_IND
P-TOKEN-GIVE	P_TOKENGIVE_REQ/P_TOKENGIVE_IND
P-TOKEN-PLEASE	P_TOKENPLEASE_REQ/P_TOKENPLEASE_IND

APLI service primitives are sent and received using calls to the functions `ap_snd()` and `ap_rcv()`. Figure 2-3 illustrates how these functions are used in combination with the APLI service primitives corresponding to the A-ASSOCIATE service to establish an association. This is a confirmed operation; therefore, there are four APLI service primitives required by the handshaking mechanism.

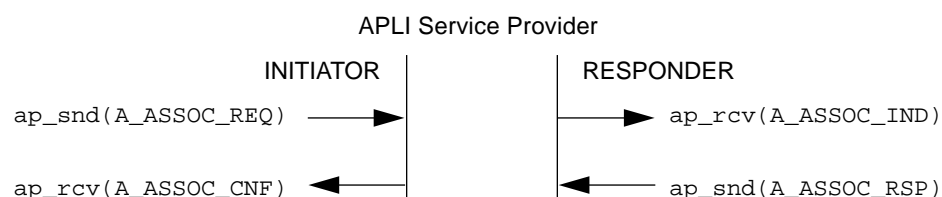


Figure 2-3 Establishing an Association

See also “`ap_snd()`—send primitive” on page 37 and “`ap_rcv()`—receive primitive” on page 40.

Sending Protocol Information

In the SunLink implementation of APLI, the union of structures **ap_cdata_t** is used to pass protocol information associated with an APLI service primitive. This union is defined in the header file `<ap_lib.h>`. It is a collation of all the possible parameters associated with the service primitives supported by APLI. Consequently, the use of the structures contained in **ap_cdata_t** is entirely dependent on the parameters defined for the specific service primitive being sent or received.

The members of the union **ap_cdata_t** that are supported by the SunLink implementation of APLI are:

```
typedef union {
    a_assoc_req_cd_t      assoc_req;
    a_assoc_ind_cd_t      assoc_ind;
    a_assoc_rsp_cd_t      assoc_rsp;
    a_assoc_cnf_cd_t      assoc_cnf;
    a_abort_req_cd_t      abort_req;
    a_abort_ind_cd_t      abort_ind;
    a_pabort_req_cd_t      pabort_req;
    a_pabort_ind_cd_t      pabort_ind;
    a_release_req_cd_t     release_req;
    a_release_ind_cd_t     release_ind;
    a_release_rsp_cd_t     release_rsp;
    a_release_cnf_cd_t     release_cnf;
    p_data_req_cd_t        data_req;
    p_tokengive_req_cd_t    tokengive_req;
    p_tokengive_ind_cd_t    tokengive_ind;
    p_tokenplease_req_cd_t  tokenplease_req;
    p_tokenplease_ind_cd_t  tokenplease_ind;
} ap_cdata_t;
```

The component structures of the union **ap_cdata_t** are specified in the *manual page* references contained in Chapter 5, “APLI Service Primitives.”

When protocol information is returned by `ap_rcv()`, APLI may have allocated data for substructures associated with the primitive. Once the service user has processed the data in the structure, the memory allocated to the substructures should be freed using the `ap_free()` function (see page 43).

Sending User Data

Almost all APLI service primitives accept a *user data* parameter. For OSI protocols, the content of user data is usually a protocol data unit (PDU) for a higher layer protocol. For example, the FTAM protocol includes an F-INITIALIZE PDU in an A-ASSOCIATE request by passing it to ACSE as the user data parameter.

All user data is passed between APLI and the APLI user in encoded form, using an appropriate transfer syntax. Unlike other parameters passed with an APLI primitive, the service user is responsible for all user data encoding and decoding, including the encoding which the ACSE and presentation protocols require for including user data in their PDUs.

The service user may choose to send the encoded data associated with a particular primitive using multiple `ap_snd()` calls. In this case, the `AP_MORE` bit in the *flags* parameter of the `ap_snd()` function must be set to indicate that part of the primitive remains to be sent. The service user may continue sending data associated with the current primitive by repeatedly calling `ap_snd()` with the `AP_MORE` bit set. Multiple calls to `ap_rcv()` are required to receive data sent using multiple calls to `ap_snd()`.

The *sptype* argument, which is used to identify the primitive to be sent, must remain the same for each `ap_snd()` call that has the `AP_MORE` bit set. Note that the value of the *sptype* argument must also be checked after each `ap_rcv()` call because another primitive (for example, an `A_PABORT_IND`) could arrive before all the data associated with the first primitive is processed.

When the last (or only) buffer of data associated with the primitive is sent, `ap_snd()` must be invoked with the `AP_MORE` bit reset. Primitives that are not associated with either an ACSE or presentation layer PDU (for example, `P_DATA_REQ`, `P_TOKENGIVE_REQ`, `P_TOKENPLEASE_REQ`) cannot be terminated by an `ap_snd()` that does not carry any user data; therefore, the final `ap_snd()` invocation must always carry one or more octets of user data for these primitives.

The service user is responsible for freeing buffers used by the `ap_snd()` and `ap_rcv()` functions. They cannot be released using `ap_free()`.

Setting the Execution Mode

An association can be operated in either *blocking* or *non-blocking* execution mode. The execution mode is either selected when the APLI instance is opened, or is selected for a specific association by setting or resetting the `AP_NDELAY` bit of the `AP_STREAM_FLAGS` attribute.

In *blocking* mode, `ap_snd()` blocks until the entire primitive has been sent and `ap_rcv()` blocks until either the entire primitive is received, or the buffer used to receive user data is filled. In the latter case, the `AP_MORE` bit of the *flags* argument is set when `ap_rcv()` returns. To receive the remainder of the primitive, the service user must continue calling `ap_rcv()` until all of the data is received and the function returns with the `AP_MORE` bit reset.

In *non-blocking* mode, `ap_snd()` and `ap_rcv()` never block. As much of the primitive is sent as possible, and `ap_rcv()` continues to read data until either the entire primitive is received, or the buffer used to receive user data is filled. In the latter case, an `AP_AGAIN` error is issued and the `AP_MORE` bit of the *flags* argument is set to indicate that a primitive was partially received. To complete sending the primitive, `ap_snd()` is called with the same set of arguments and `ap_rcv()` is called to receive the data. When the entire primitive has been received, the `AP_MORE` bit is reset.

The function `ap_poll()` can be used to check on the availability of resources by showing the activity of the APLI instance when sending or receiving primitives using multiple calls to `ap_snd()` and `ap_rcv()`.

Closing an APLI Instance

An APLI instance is closed by calling the function `ap_close()` with the file descriptor (*fd*) that was returned by `ap_open()` when the instance was opened. This terminates the APLI instance and releases the memory that was originally allocated to the APLI environment. The service user is responsible for ensuring that any dynamically allocated memory has been released using the `ap_free()` function (see page 43).

The function `ap_close()` does not necessarily close the STREAM identified by the file descriptor (*fd*). In the event that the STREAM is shared by more than one APLI instance (each with a different file descriptor), it remains open until the last instance is closed.

See also “`ap_close()`—close APLI instance” on page 49.

Setting the Token Assignment

The token assignment used during association establishment and during resynchronization are formed by OR'ing together bits corresponding to the requested position for each token. Assignments for unavailable tokens are ignored. If no assignment is given for an available token, `AP_service_TOK_REQ` is specified by the service.

The association-initiator may set the following values:

Table 2-2 Tokens Set by Requestor

Token	Allowed Values
data	AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE
synchronize-minor	AP_SYNCMINOR_TOK_REQ AP_SYNCMINOR_TOK_ACPT AP_SYNCMINOR_TOK_CHOICE
major/activity	AP_MAJACT_TOK_REQ AP_MAJACT_TOK_ACPT AP_MAJACT_TOK_CHOICE
release	AP_RELEASE_TOK_REQ AP_RELEASE_TOK_ACPT AP_RELEASE_TOK_CHOICE

The association-responder may set the following values for those tokens identified as “responder’s choice” by the association-initiator:

Table 2-3 Tokens Set by Acceptor

Token	Allowed Values
data	AP_DATA_TOK_REQ AP_DATA_TOK_ACPT
synchronize-minor	AP_SYNCMINOR_TOK_REQ AP_SYNCMINOR_TOK_ACPT
major/activity	AP_MAJACT_TOK_REQ AP_MAJACT_TOK_ACPT
release	AP_RELEASE_TOK_REQ AP_RELEASE_TOK_ACPT

Housekeeping and Auxiliary Functions

In addition to the basic functions that access the ACSE and presentation layer services, APLI provides a number of functions that manage the operation of the APLI instance.

Releasing Allocated Memory

Many of the data structures used by the APLI functions have pointers to structures that are allocated dynamically. If the memory allocated to these structures is not released periodically by calling the function `ap_free()`, it will continue to grow throughout the life of the APLI instance. The function `ap_free()` is typically used to release memory allocated as the result of a call to `ap_get_env()`.

See also “`ap_free()`—free allocated memory” on page 43.

Polling the APLI Instance

The function `ap_poll()` is a macro for the UNIX system call `poll(2)`. It is used to check the activity of the APLI instance identified by the file descriptor *fd*. The use of `ap_poll()` is not affected by whether the association is operated in *blocking* or *non-blocking* mode.

The function `ap_poll()` is typically used in conjunction with the `AP_MORE` bit to send a single primitive using multiple calls to `ap_snd()`, or in conjunction with `ap_rcv()` to wait for incoming events..

See also “`ap_poll()`—poll APLI events” on page 45.

Reporting Errors

Almost all APLI functions return a result code as the function value. The result is 0 if the function was completely successful, and -1 if any error or warning condition occurred. When a function returns a result of -1, the global variable `ap_errno` is set to indicate the source of the error or warning.

The error message string that corresponds to a specific error code is returned by calling the function `ap_errno()`.

See also “`ap_error()`—return error message” on page 47.

Error Summary

A complete list of the errors that are reported by the SunLink implementation of APLI is contained in Table 2-4.

Table 2-4 APLI Error Codes

Error Code	Description
AP_ACCES	Request to bind to specified address denied.
AP_AGAIN	Request not completed.
AP_BADATTRVAL	Bad value for environment attribute.
AP_BADCD_ACT_ID	Cdata field value invalid: act_id.
AP_BADCD_DIAG	Cdata field value invalid: diag.
AP_BADCD_EVT	Cdata field value invalid: event.
AP_BADCD_OLD_ACT_ID	Cdata field value invalid: old_act_id.
AP_BADCD_OLD_CONN_ID	Cdata field value invalid: old_conn_id.
AP_BADCD_PABORT_IND	Cdata field value invalid: pabort_ind.
AP_BADCD_RES	Cdata field value invalid: res.
AP_BADCD_RES_SRC	Cdata field value invalid: res_src
AP_BADCD_RESYNC_TYPE	Cdata field value invalid: resync_type.
AP_BADCD_RSN	Cdata field value invalid: rsn.
AP_BADCD_SRC	Cdata field value invalid: src.
AP_BADCD_SYNC_P_SN	Cdata field value invalid: sync_p_sn.
AP_BADCD_SYNC_TYPE	Cdata field value invalid: sync_type.
AP_BADCD_TOKENS	Cdata field value invalid: tokens.
AP_BADENC	Bad encoding choice in enveloping function.
AP_BADENV	A mandatory attribute is not set.
AP_BADF	Not a presentation service endpoint.
AP_BADFLAGS	Combination of flags is invalid.
AP_BADFREE	Could not free structure members.
AP_BADKIND	Unknown structure type.
AP_BADLSTATE	Instance in bad state for that command.

Table 2-4 APLI Error Codes

Error Code	Description
AP_BADPARSE	Attribute parse failed.
AP_BADPRIM	Unrecognized primitive from user.
AP_BADREF	Bad reference in enveloping function.
AP_BADRESTR	Not restored due to more bit on (AP_MORE set).
AP_BADROLE	Request invalid due to value of AP_ROLE.
AP_BADSAVE	Not saved due to more bit on (AP_MORE set).
AP_BADSAVEF	Invalid FILE pointer.
AP_BADUBUF	Bad length for user data.
AP_HANGUP	Association closed or aborted.
AP_INTERNAL	Internal error.
AP_LOOK	A pending event requires attention.
AP_NOATTR	No such attribute.
AP_NOENV	No environment for that <i>fd</i> .
AP_NOERROR	No error.
AP_NOMEM	Could not allocate enough memory.
AP_NOREAD	Attribute is not readable.
AP_NOSET	Attribute is not settable.
AP_NOWRITE	Attribute is not writable.
AP_PDUREJ	Invalid PDU rejected.

Compiling and Linking APLI Applications

Applications developed using the services provided by APLI must be compiled and linked with the following runtime libraries and include files:

Runtime Libraries

Applications developed with APLI must be linked with the following runtime libraries:

`libapli.a` provides the direct services defined by APLI.

`libapli.so.1` provides descriptions of the APLI data structures.

These libraries are delivered with the SunLink OSI Communication Platform. By default, they are installed under `/opt/SUNWconn/osinet/lib`. If you changed the default base directory when you installed the SunLink OSI Communication Platform, these libraries will be located under `<basedir>/SUNWconn/osinet/lib`.

External Include Files

You must include the following external include files in applications developed using APLI:

`ap_lib.h` contains constant, data structure, attribute, and error message definitions.

`osi_lib.h` contains additional constant and data structure definitions (for example, `osi_buf`, `any_t`).

If you want to use the function `ap_poll()` to poll an APLI instance, you must also include the following external include file:

`ap_poll.h` contains poll constant and data structure definitions.

These external include files are delivered with the SunLink OSI Communication Platform. By default, they are installed under `/opt/SUNWconn/osinet/include`. If you changed the default base directory when you installed the SunLink OSI Communication Platform, these libraries will be located under `<basedir>/SUNWconn/osinet/lib`.

Compile and Link Procedure

To compile and link applications developed with APLI, you must specify the location of the include files and runtime libraries by incorporating the following lines (or similar) in your Makefile:

```
CFLAGS += -I /opt/SUNWconn/osinet/include
LD_FLAGS += -L /opt/SUNWconn/osinet/lib
```

Alternatively, you can specify the location of the include files and runtime libraries as illustrated in the example Makefile located in the directory `/opt/SUNWconn/osinet/example`.

<i>APLI Function Summary</i>	<i>page 28</i>
<i>ap_open()—open APLI instance</i>	<i>page 29</i>
<i>ap_init_env()—initialize APLI environment attributes</i>	<i>page 31</i>
<i>ap_get_env()—get APLI environment attribute</i>	<i>page 33</i>
<i>ap_set_env()—set APLI environment attribute</i>	<i>page 35</i>
<i>ap_snd()—send primitive</i>	<i>page 37</i>
<i>ap_rcv()—receive primitive</i>	<i>page 40</i>
<i>ap_free()—free allocated memory</i>	<i>page 43</i>
<i>ap_poll()—poll APLI events</i>	<i>page 45</i>
<i>ap_error()—return error message</i>	<i>page 47</i>
<i>ap_close()—close APLI instance</i>	<i>page 49</i>

This chapter defines the functions provided by the SunLink implementation of APLI. The function definitions are presented in the form of *manual pages* that provide detailed specifications of parameters and structures.

Code fragments that are included in each manual page to illustrate how the functions could be used as part of a custom application. They are not intended to represent compilable code, and should not be construed as an attempt to provide a template for constructing any particular application. A compilable example application is provided with the Sunlink OSI Communication Platform and the files associated with this example are located in the directory

/opt/SUNWconn/osinet/example. See Appendix A, “APLI Example Application” for a detailed description of how to compile and run this example application.

APLI Function Summary

The functions provided by the SunLink implementation of APLI are listed in Table 3-1.

Table 3-1 APLI Function Summary

Function	Description	Reference
ap_open()	Open APLI instance	page 29
ap_init_env()	Initialize APLI environment	page 31
ap_get_env()	Get APLI environment attribute	page 33
ap_set_env()	Set APLI environment attribute	page 35
ap_snd()	Send APLI service primitive	page 37
ap_rcv()	Receive APLI service primitive	page 40
ap_free()	Free memory for APLI data structures	page 43
ap_poll()	Poll APLI resources	page 45
ap_error()	Return an error message	page 47
ap_close()	Close APLI instance	page 49

ap_open () —*open APLI instance*

Name

ap_open—open a new APLI instance.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_open (
    char *pathname,          /* ptr to device */
    int  oflags              /* set execution mode */
);
```

Description

This function creates a new instance of APLI that uses the service provider specified by *pathname*. In the case of the SunLink implementation of APLI, *pathname* must identify the device `/dev/oopi` that corresponds to the SunLink OSI Communication Platform. Note that *pathname* may be the actual device name `/dev/oopi`, or a symbolic link that points to this device.

The Sunlink implementation of APLI will not accept a `NULL` value for *pathname*.

The *oflags* argument is a bit mask set to `O_NDELAY` to request non-blocking execution mode (no delay). Unless specified otherwise, the APLI instance operates in blocking mode.

The APLI environment must be initialized using a call to the `ap_init_env()` function before the APLI instance can be used to send or receive primitives.

Return Value

On successful completion, `ap_open()` returns the **integer** value file descriptor (*fd*) that is used to identify the service provider in all subsequent function calls. Otherwise, a value of `-1` is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

AP_NOMEM

In addition, **operating system** class errors are reported.

Example

Code Fragment 3-1 shows how `ap_open()` can be used to open an APLI instance that will operate in blocking mode (default) by setting *oflags* to NULL or zero. To operate in non-blocking mode, *oflags* must be set to `O_NDELAY` (no delay).

Code Fragment 3-1 Using `ap_open()`

```
#include <ap_lib.h>
#include <osi_lib.h>

int OpenApliInst ()
{
    int fd;
    char *descName="/dev/oopi";

    if ((fd = ap_open(descName, NULL)) == FAIL) {
        printf ("Error in ap_open %s \n", ap_error());
        return (FAIL)
    }
    return (fd);
}
```


`ap_init_env()` — *initialize APLI environment attributes*

Name

`ap_init_env`—establish and initialize a single instance of APLI.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_init_env (
    int fd,                      /* file descriptor */
    const char *env_file,        /* ignored */
    int flags                    /* unused */
);
```

Description

This function initializes the APLI environment for the service provider (`/dev/ooip`) identified by the file descriptor `fd`. The argument `flags` is unused and the pointer `env_file` is ignored by the SunLink implementation of APLI. The `env_file` elements are always set to their default values unless changed dynamically using the function `ap_set_env()`.

This function must be called to initialize each APLI instance created by calling the `ap_open()` function (see page 43), which returns the file descriptor `fd`. Calling `ap_init_env()` allocates memory for the environment attributes and sets the environment attributes to their default values.

See Chapter 4, “APLI Environment Attributes” for a detailed description of the APLI environment including the default values allocated to each attribute.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of `-1` is returned and `ap_errno` is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_BADF`, `AP_NOATTR`, `AP_ACCESS`, `AP_NOWRITE`, `AP_NOMEM`

Example

Code Fragment 3-2 shows how `ap_init_env()` can be used to initialize the environment of an APLI instance identified by the file descriptor *fd*.

Code Fragment 3-2 Using `ap_init_env()`

```
#include <ap_lib.h>
#include <osi_lib.h>

int InitApliEnv (fd)
{
    if (ap_init_env(fd, NULL, 0) != SUCCESS) {
        printf ("Error in ap_init_env %s \n", ap_error());
        return (FAIL);
    }
    return (SUCCESS);
}
```

`ap_get_env()` —get APLI environment attribute

Name

`ap_get_env`—get the value of an APLI environment attribute.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_get_env (
    int fd,                /* file descriptor */
    unsigned long attr,    /* attribute name */
    void *val              /* ptr to attribute */
);
```

Description

This function retrieves the value of the environment attribute *attr* for the service provider (`/dev/ooopi`) identified by the file descriptor *fd*. Valid attribute names are listed in the “APLI Attribute Summary” on page 52.

The value supplied to this function as the *val* argument depends on which attribute is to be examined. In all cases, *val* must point to an object of the same type as the specified attribute.

For example, if the type of the attribute is **long**, *val* must point to a **long**. Similarly, if the type of the attribute is **ap_dcs_t**, *val* must point to an **ap_dcs_t** structure.

If the object pointed to by *val* is either a pointer or a structure that includes pointers (for example, **ap_dcs_t**), APLI allocates additional memory and assigns proper values to the pointer elements as required. The memory allocated to the environment attributes by `ap_get_env()` can be released by calling the `ap_free()` function (see page 43).

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

AP_NOATTR, AP_NOENV, AP_NOREAD, AP_NOMEM

In addition, **operating system** class errors may be reported.

Example

Code Fragment 3-3 shows how `ap_get_env()` can be used to return the current state `AP_STATE` and check for the state `AP_IDLE`.

Code Fragment 3-3 Using `ap_get_env()`

```
#include <ap_lib.h>
#include <osi_lib.h>

int GetApliState ()
{
    int          rc;           /* return code from ap_get_env */
    unsigned long state;       /* current state */

    rc=ap_get_env(fd, AP_STATE, &state);
    if ((rc != 0) && (state != AP_IDLE)) {
        printf ("Error in ap_get_env %s \n", ap_error());
        return (FAIL);
    }
    return (SUCCESS);
}
```

`ap_set_env()` —set APLI environment attribute

Name

`ap_set_env()`—set APLI environment

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_set_env (
    int fd,                /* file descriptor */
    unsigned long attr,    /* attribute name */
    ap_val_t val           /* value assigned to attr */
);
```

Description

This function sets or resets the value of the environment attribute *attr* for the service provider (`/dev/ooopi`) identified by the file descriptor *fd*. Valid attribute names are listed in “APLI Attribute Summary” on page 52.

The *val* argument is a union that is used to pass the value that is to be assigned to the specified attribute. The *val* union is defined as:

```
typedef union {
    long l;
    void *v
} ap_val_t
```

In all cases, *val* must be of the same type as the attribute to be modified.

For example, if the value of the attribute that is to be modified is an integer or bit mask, the *l* member of the **ap_val_t** union must contain a **long** or **unsigned long**. Otherwise, the *v* member of the **ap_val_t** union must contain a pointer to a structure of the same type as the specified attribute.

Refer to Chapter 4, “APLI Environment Attributes” for a detailed description of the attribute types.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the *ap_error()* function (see page 47).

Errors

AP_BADATTRVAL, *AP_NOATTR*, *AP_NOENV*, *AP_NOMEM*, *AP_NOWRITE*, *AP_HANGUP*, *AP_ACCESS*

In addition, **operating system** class errors may be reported.

Example

Code Fragment 3-4 shows how *ap_set_env()* can be used to set the *AP_INITIATOR* bit of the *AP_ROLE_ALLOWED* attribute.

Code Fragment 3-4 Using *ap_set_env()*

```
#include <ap_lib.h>
#include <osi_lib.h>

int SetApliRole ()
{
    ap_val_t val;                /* value passed to ap_set_env */
    val.l = AP_INITIATOR         /* set l member of ap_val_t */

    if ((ap_set_env(fd, AP_ROLE_ALLOWED, val))) {
        printf ("Error in ap_set_env %s \n", ap_error());
        return (FAIL);
    }
    return (SUCCESS);
}
```

ap_snd() —*send primitive*

Name

ap_snd()—send ACSE/presentation service primitive

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,          /* file descriptor */
    unsigned long sptype,     /* primitive to be sent */
    ap_cdata_t   *cdata,     /* union for protocol info */
    struct osi_buf *ubuf,     /* buffer for sending data */
    int flags       /* set to indicate pending data */
);
```

Description

This function is used to send a request (REQ) or response (RSP) primitive. The file descriptor *fd* points to the service provider (/dev/oopi). The *sptype* parameter contains a symbolic constant that identifies the primitive to be sent. The valid symbolic constants listed in Chapter 5, “APLI Service Primitives” are derived from the primitive names by prefixing AP_ to the name.

Any additional protocol information to be sent with a primitive is conveyed by the union of structures **ap_cdata_t** pointed to by *cdata*. The value *sptype* defines which member of the union **ap_cdata_t** is affected.

Any encoded (ISO 8650) user data associated with the primitive is sent in the *osi_buf* structure pointed to by *ubuf*. The *osi_buf* structure has the following members:

```
unsigned int maxlen      /* maximum buffer length */
unsigned int len         /* length of data in buffer */
char *buf               /* ptr to data buffer */
```

Before issuing the `ap_snd()` call, `ubuf→buf` must point to the buffer where the data is to be placed and `ubuf→len` must be set to the number of bytes of user data in the buffer. If no user data is to be sent, `ubuf` must be `NULL`.

The `flags` argument must point to an **integer** when `ap_snd()` is called. The `AP_MORE` bit of `flags` is set to indicate that repeated `ap_snd()` calls are required to send a single primitive. Each `ap_snd()` call with `AP_MORE` set indicates that another `ap_snd()` will follow. An `ap_snd()` call with `AP_MORE` reset indicates that the primitive is complete. The value of the `sptype` argument must be the same for all `ap_snd()` calls used to send a single primitive.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `ap_errno` is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_BADENV`, `AP_BADLSTATE`, `AP_BADPRIM`,
`AP_BADUBUF`, `APHANGUP`, `AP_NOENV`

In addition, **operating system, asn.1, acse, presentation, session and transport** class errors may be reported.

Example

Code Fragment 3-5 on page 39 shows how `ap_snd()` can be used to issue an association request by sending the `AP_ASSOC_REQ` primitive.

Code Fragment 3-5 Using ap_snd()

```
#include <ap_lib.h>
#include <osi_lib.h>

int SndApliAReq ()
{
    ap_cdata_t      cdata;
    struct osi_buf  ubuf;
    unsigned long   sptype;
    int flags;
    int rc;
    char msg[128];

    cdata.assoc_req.tokens =
        AP_DATA_TOK_REQ | AP_SYNCMINOR_TOK_REQ |
        AP_MAJACT_TOK_REQ | AP_RELEASE_TOK_REQ;
    cdata.assoc_req.udata_length = 0;

    flags =      0      /* No AP_MORE bit set, prim sent in one call */
    ubuf.maxlen =128;
    ubuf.len =    0;
    ubuf.buf =    msg;

    rc = ap_snd(fd, A_ASSOC_REQ, cdata, &ubuf, flags);
    if (rc != 0) {
        printf("Error in ap_snd A_ASSOC_REQ %s \n", ap_error());
        return(FAIL)
    }
    return (SUCCESS);
}
```

ap_rcv() —receive primitive

Name

ap_rcv—receive an ACSE/Presentation primitive.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,          /* file descriptor */
    unsigned long *sptype,    /* ptr to primitive name */
    ap_cdata_t   *cdata,     /* union for protocol info */
    struct osi_buf*ubuf,     /* buffer to receive data */
    int *flags              /* indicates pending data */
);
```

Description

This function is used to receive indication (IND) or confirmation (CNF) primitives. The file descriptor *fd* points to the service provider (/dev/oopi).

When `ap_rcv()` is called, *sptype* must point to an **unsigned long**, and *cdata* must point to the **ap_cdata_t** union defined in the header file `<ap_lib.h>`.

Upon return, the value of the **unsigned long** pointed to by *sptype* contains a symbolic constant that identifies the received primitive. The valid symbolic constants are derived from the primitive names by prefixing the name with `AP_` and are defined in Chapter 5, “APLI Service Primitives.”

Any additional protocol information received with a primitive is conveyed by the union of structures **ap_cdata_t** pointed to by *cdata*. The value returned in *sptype* defines which members of the union **ap_cdata_t** are affected.

Any encoded (**ISO 8650**) user data associated with the received primitive will be returned to the user in the *osi_buf* structure pointed to by *ubuf*. The *osi_buf* structure has the following members:

```
unsigned int maxlen      /* maximum buffer length */
unsigned int len         /* length of data in buffer */
char *buf               /* ptr to data buffer */
```

Before issuing the `ap_rcv()` call, *ubuf*→*buf* must point to the buffer where the data is to be placed and *ubuf*→*maxlen* must be set to the size of the buffer. APLI will copy data to the buffer beginning at the location (*ubuf*→*buf*)+(ubuf→len). On return, *ubuf*→*len* will be incremented by the amount of data written to the buffer.

The *flags* argument must point to an **integer** when `ap_rcv()` is called. On return, and if all of the data has not been received, the `AP_MORE` bit of *flags* will be set. This indicates that repeated `ap_rcv()` calls are required to complete the data transfer.

The *sptype* argument must be checked after each `ap_rcv()` call because a new primitive (for example, `A_PABORT_IND`) may arrive before all of the data is received and the remaining data will be lost. If APLI is operating in synchronous mode, `ap_rcv()` blocks until either an entire primitive is received or the buffer is filled. If APLI is operating in asynchronous mode, `ap_rcv()` returns an `AP_AGAIN` error, which signals that the operation should be repeated.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_PDUREJ`, `AP_BADLSTATE`, `AP_NOBUF`

In addition, **operating system** and **asn.1** class errors may be reported.

Example

Code Fragment 3-6 shows how `ap_rcv()` can be used to receive an association indication.

Code Fragment 3-6 Using `ap_rcv()`

```
#include <ap_lib.h>
#include <osi_lib.h>

Int RcvApliInd ()
{
    ap_cdata_t    cdata;
    struct osi_buf ubuf;
    unsigned long sptype;
    int flags;
    int rc;
    char msg[128];

    ubuf.maxlen = 128;
    ubuf.len     = 0;
    ubuf.buf     = msg;

    rc = ap_rcv(fd, &sptype, &cdata, &ubuf, &flags);
    if (rc != 0) {
        printf("Error in ap_rcv A_ASSOC_IND %s \n", ap_error());
        return(FAIL);
    }
    return (SUCCESS);
}
```

ap_free() — *free allocated memory*

Name

ap_free—free user memory associated with APLI data structures.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_free (
    unsigned long kind,      /* type of structure */
    void *val                /* ptr to structure of type kind */
);
```

Description

This function frees memory allocated for structures used to convey the values of APLI environment attributes, or *ap_cdata_t* structures that are associated with APLI primitives. Many of these structures have pointers to variables that are allocated dynamically. If these structures are not freed, the memory they occupy will continue to grow throughout the life of the APLI instance. A typical use of this function is to free memory allocated for the value of an environment attribute following an *ap_get_env()* invocation. Note that *ap_free()* cannot be used to free user-allocated memory.

The argument *kind* identifies the kind of structure that is to be freed. Legal values for this argument are:

- The names of APLI environment attributes listed in the “Attribute” column of the environment attribute summary given in “Environment Data Types” on page 72 (for example, *AP_INIT_TOKENS*).
- The names of APLI service primitives.
- The names of types associated with either APLI environment attributes or service primitives, in capital letters.

The argument *val* is a pointer to a structure of the type indicated by *kind*. When releasing memory, *ap_free()* follows and frees all internal pointers. The top level structure (the structure pointed to by *val*) is not freed. If the specified structure does not contain any pointers, *ap_free()* has no effect.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the *ap_error()* function (see page 47).

Errors

AP_BADFREE, AP_BADKIND

Example

Code Fragment 3-7 shows how memory allocated to the AP_CDL environment attribute by *ap_get_env()* can be released by calling *ap_free()* with *kind* set to AP_CDL (or AP_CDL_T) and *val* pointing to the relevant **ap_cdl_t** structure.

Code Fragment 3-7 Using ap_free()

```
/* proposed presentation context definition list */
ap_cdl_t *pcdl

/* get proposed pcdl from the environment */
if (ap_get_env (fd, AP_CDL, *pcdl) != SUCCESS)
{
    printf ("ap_get_env AP_CDL %s \n", ap_error());
}
.
.
.

/* free memory allocated to pcdl by ap_get_env */
ap_free(AP_CDL, *pcdl);
```

ap_poll() — *poll APLI events*

Name

ap_poll—poll for the occurrence of specified APLI events.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_poll (
    struct ap_pollfd *fds, /* array of nfds structures */
    int nfds,              /* size of array */
    int timeout            /* delay before timeout */
);
```

Description

This function provides a consistent interface for detecting when certain events have occurred on an APLI instance. It is a macro for the UNIX system call `poll(2)`.

The *fds* argument is an array of *nfds* **ap_pollfd_t** structures defined as:

```
int fd;                /* APLI instance identifier */
short events;          /* requested events */
short revents;         /* returned events */
```

The argument *fd* should correspond to an APLI communication endpoint; however, any open file descriptor is accepted. The *events* field is a bitmask used to indicate which events should be reported for the instance. The *revents* field will be set by APLI to indicate which of the requested events have occurred:

Event Code	Description
AP_POLLIN	Data has arrived (on either band) and is available to be read.
AP_POLLOUT	Data can be sent on the normal priority band.

If none of the defined events have occurred on the selected instances, `ap_poll()` waits *timeout* milliseconds for an event to occur on one of the selected instances before returning. If the value of *timeout* is `AP_INFTIM`, `ap_poll()` waits until a requested event occurs or until the call is interrupted.

The `ap_poll()` call is not affected by whether the APLI instance is operating in blocking or non-blocking execution mode.

Return Value

Upon successful completion, a non-negative value is returned that indicates the number of instances for which *revents* is non-zero. A return value of 0 indicates that the call timed out and no instances were selected. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

Only **operating system** class errors may be reported.

Example

Code Fragment 3-8 shows how `ap_poll()` can be used wait for the occurrence of a certain event.

Code Fragment 3-8 Using `ap_poll()`

```
struct poll_fdpoll_fd;
struct timevaltimeout;

poll_fd.fd = fd;
poll_fd.events = AP_POLLIN;
while ((rc = ap_poll(&poll_fd,1,&timeout)) > 0)
{
    .
    .
    .
}
```


`ap_error()` —*return error message*

Name

ap_error—return an error message.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

char *ap_error ( )           /* ptr to error message */
```

Description

This function returns a pointer to a message that corresponds to the error code returned in *ap_errno*. The error code *ap_errno* is set each time a function returns with a value of `-1`.

The message is in the natural language of the currently defined `LOCALE`. The pointer will point to `NULL` if no such message is available under the currently defined `LOCALE`. For English language locales, the message must be one of the messages listed in “Error Summary” on page 23.

All error codes that are not APLI errors (and thus do not map to error strings) will return a generic error string.

The message pointer points to an internal buffer area that is overwritten by the next call to `ap_error()`. To retain the message text, it should be copied to some private storage. A successful function call, does not overwrite the buffer.

Note that `ap_error()` is set to `AP_LOOK` in the event that a transfer is interrupted before all the data has been sent and received. This does not necessarily indicate an error condition; however, it indicates that an event is pending that requires attention.

Return Value

Upon completion, a pointer to the appropriate error message is returned.

Errors

Never fails. No error conditions are reported by this function.

Example

Code Fragment 3-9 shows how `ap_error()` can be used to return an error message after calling an APLI function.

Code Fragment 3-9 Using `ap_error()`

```
#include <ap_lib.h>
#include <osi_lib.h>

int InitApliEnv (fd)
{
    if (ap_init_env(fd, NULL, 0) != SUCCESS) {
        printf ("Error in ap_init_env %s \n", ap_error());
        return (FAIL);
    }
    return (SUCCESS);
}
```

`ap_close()` —close APLI instance

Name

`ap_close`—close an APLI instance.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_close (
    int fd                /* file descriptor */
);
```

Description

This function frees the resources allocated to support the service provider identified by the file descriptor *fd*. It must be called to terminate each APLI instance created using the `ap_open()` function (see page 43).

If the STREAM identified by *fd* is shared by more than one APLI instance, it remains open until the last open file descriptor is closed. If the APLI instance is closed while an association is still active, the association is aborted before the resources are released.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the cause of error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

AP_BADF

In addition, **operating system** class errors may be reported.

Example

Code Fragment 3-10 Using `ap_close()`

```
#include <ap_lib.h>
#include <osi_lib.h>

int CloseApli ()
{
    int fd

    if (ap_close(fd) != SUCCESS) {
        printf ("Error in ap_close %s \n", ap_error());
        return (FAIL);
    }
    return (SUCCESS);
}
```

APLI Environment Attributes



<i>APLI Attribute Summary</i>	<i>page 52</i>
<i>Environment Data Types</i>	<i>page 72</i>
<i>Environment Read/Write Table</i>	<i>page 81</i>

The APLI environment is made up of a set of attributes that are used to keep state information and to hold the various pieces of data required to establish and maintain an association with another application entity. This chapter describes the environment attributes that are supported by the SunLink implementation of APLI.

The current value assigned to a given attribute is returned by calling the function `ap_get_env()`; the value assigned to a given attribute is modified by calling the function `ap_set_env()`.

Most of the APLI attributes have default values which are set when the APLI instance is initialized by calling the function `ap_init_env()` and in many cases these default values will be sufficient to allow a service user to access ACSE and presentation layer services. However, there are a minimum number of environment attributes that must be modified before an association can be established.

Before initiating an association request, the service user must ensure that the following environment attributes are set correctly:

AP_BIND_PADDR, AP_CNTX_NAME, AP_REM_PADDR, AP_LIB_SEL

Before responding to an association request, the service user must ensure that the following environment attributes are set correctly:

AP_BIND_PADDR, AP_LIB_SEL

Note that there are no default values for AP_BIND_PADDR and AP_REM_PADDR.

APLI Attribute Summary

The environment attributes supported by the SunLink implementation of APLI are listed in Table 4-1:

Table 4-1 Attribute Summary

Attribute	Description
AP_ACSE_AVAIL	Indicates the available versions of the ACSE protocol
AP_ACSE_SEL	Indicates the selected version of the ACSE protocol
AP_BIND_PADDR	Binds the local presentation address to an APLI instance
AP_CLD_AEID	Conveys the called <i>AE-invocation-id</i> of the AARQ APDU
AP_CLD_AEQ	Conveys the called <i>AE-qualifier</i> of the AARQ APDU
AP_CLD_APID	Conveys the called <i>AP-invocation-id</i> of the AARQ APDU
AP_CLD_APT	Conveys the called <i>AP-title</i> of the AARQ APDU
AP_CLD_CONN_ID	Conveys the <i>session-connection-id</i> proposed by the responder
AP_CLG_AEID	Conveys the calling <i>AE-invocation-id</i> of the AARQ APDU
AP_CLG_AEQ	Conveys the calling <i>AE-qualifier</i> of the AARQ APDU
AP_CLG_APID	Conveys the calling <i>AP-invocation-id</i> of the AARQ APDU
AP_CLG_APT	Conveys the calling <i>AP-title</i> of the AARQ APDU
AP_CLG_CONN_ID	Conveys the <i>session-connection-id</i> proposed by the initiator
AP_CNTX_NAME	Conveys the <i>application-context-name</i> of AARQ and AARE APDUs
AP_DCS	Specifies the <i>defined-context-set</i> of abstract/transfer syntax pairs
AP_DCN	Indicates the <i>default-context-name</i> of the CP PPDU

Table 4-1 Attribute Summary

Attribute	Description
AP_DPCR	Indicates the <i>default-context-result</i> of the CPR PPDU
AP_STREAM_FLAGS	Indicates the execution mode of the APLI instance
AP_INIT_SYNC_PT	Conveys the initial session synchronization point serial number
AP_LCL_PADDR	Indicates the local presentation address
AP_LIB_AVAIL	Indicates the available versions of the APLI libraries
AP_LIB_SEL	Indicates the selected version of the APLI libraries
AP_MODE_AVAIL	Indicates the available modes of operation
AP_MODE_SEL	Indicates the selected mode of operation
AP_MSTATE	Indicates that data is pending
AP_PCDL	Lists the <i>presentation-contexts</i> proposed by the initiator
AP_PCDRL	Lists the <i>presentation-contexts</i> accepted/rejected by responder
AP_PFU_AVAIL	Indicates which <i>presentation-functional-units</i> are available
AP_PFU_SEL	Indicates which <i>presentation-functional-units</i> are selected
AP_PRES_AVAIL	Indicates the available versions of the presentation layer
AP_PRES_SEL	Indicates the selected version of the presentation layer protocol
AP_REM_PADDR	Indicates the remote presentation address
AP_ROLE_ALLOWED	Indicates if the APLI instance can act as initiator or responder
AP_ROLE_CURRENT	Indicates if the APLI instance is acting as initiator or responder
AP_RSP_AEID	Indicates the responding <i>AE-invocation-id</i> of AARE APDU
AP_RSP_AEQ	Indicates the respondig <i>AE-qualifier</i> of the AARE APDU
AP_RSP_APID	Indicates the responding <i>AP-invocation-id</i> of the AARE APDU
AP_RSP_APT	Indicates the responding <i>AP-title</i> of the AARE APDU
AP_SESS_AVAIL	Indicates the available versions of the session layer protocol
AP_SESS_SEL	Indicates the selected session layer protocol
AP_SESS_OPT_AVAIL	Indicates which optional session capabilities are available
AP_SFU_AVAIL	Indicates which optional <i>session-functional-units</i> are available
AP_SFU_SEL	Indicates which optional <i>session-functional-units</i> are selected

Table 4-1 Attribute Summary

Attribute	Description
AP_STATE	Indicates the current state of the APLI instance
AP_TOKENS_AVAIL	Indicates which tokens are available for assignment
AP_TOKENS_OWNED	Indicates tokens currently assigned to the service user

AP_ACSE_AVAIL

Indicates the available versions of the ACSE protocol. For the SunLink implementation of APLI, this attribute always has the AP_ACSEVER1 bit set.

Type	unsigned long
Values (bit):	AP_ACSEVER1 (default)
Readable:	always
Writable:	never

AP_ACSE_SEL

Indicates the selected version of the ACSE protocol. For the SunLink implementation of APLI, this attribute always has the AP_ACSEVER1 (Upper Layer Services Release 1.1) bit set.

Type	unsigned long
Values (bit):	AP_ACSEVER1 (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_BIND_PADDR

Binds a presentation address to the APLI instance. There is no default value; therefore AP_BIND_PADDR must always be set after an APLI instance is initialized. Setting this attribute sets the attribute AP_LCL_PADDR to the same value. It is the responsibility of the service user to build a valid presentation address.

Type	structure of type ap_paddr_t (see page 78)
Values:	any valid presentation address, no default
Readable:	always
Writable:	only in states : AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_CLD_AEID

Optional parameter that conveys the called *AE-invocation-identifier* of the AARQ APDU. By default, this parameter is set to AP_CLD_AEID_NOVAL (not present).

Type	long
Values:	any valid AEID, AP_CLD_AEID_NOVAL (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_CLD_AEQ

Optional parameter that conveys the called *AE-qualifier* of the AARQ APDU. By default, this argument is not present as specified by setting the *size* argument of **any_t** to -1.

Type	structure of type any_t (see page 79)
Values:	any valid AEQ or not present (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_CLD_APID

Optional parameter that conveys the called *AP-invocation-identifier* of the AARQ APDU. By default, this attribute is set to AP_CLD_APID_NOVAL (not present).

Type	long
Values:	any valid APID, AP_CLD_APID_NOVAL (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_CLD_APT

Optional parameter that conveys the called *AP-title* of the AARQ APDU. By default, this argument is not present as indicated by setting the *size* argument of *any_t* to -1.

Type	structure of type <i>any_t</i> (see page 79)
Values:	any valid APT or not present (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_CLD_CONN_ID

Optional parameter that conveys the value of the *session-connection-identifier* that is proposed by the association-responder. By default, this attribute is set to NULL (not present).

Type	structure of type <i>ap_cld_conn_id_t</i>
Values:	any valid <i>session-connection-identifier</i> or not present (default)
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_CLG_AEID

Optional parameter that conveys the calling *AE-invocation-identifier* of the AARQ APDU. By default, this parameter is set to AP_CLG_AEID_NOVAL (not present).

Type	long
Values:	any valid AEID, AP_CLG_AEID_NOVAL (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_CLG_AEQ

Optional parameter that conveys the calling *AE-qualifier* of the AARQ APDU. By default, this attribute is not present as indicated by setting the *size* argument of **any_t** to -1.

Type	structure of type any_t (see page 79)
Values:	any valid AEQ or not present (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_CLG_APID

Indicates the calling *AP-invocation-identifier* of the AARQ APDU. By default, this attribute is set to AP_CLG_APID_NOVAL (not present).

Type	long
Values:	any valid APID, AP_CLG_APID_NOVAL (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_CLG_APT

Indicates the calling *AP-title* of the AARQ APDU. By default, this attribute is not present as indicated by setting the *size* argument of **any_t** to -1.

Type	structure of type any_t (see page 79)
Values:	any valid APT or not present (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_CLG_CONN_ID

Optional parameter that conveys the value of the *session-connection-identifier* that was proposed by the association-initiator. By default, this attribute is set to NULL (not present).

Type	structure of type ap_clg_conn_id_t (see page 75)
Values:	any valid <i>session-connection-identifier</i> or not present (default)
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE

AP_CNTX_NAME

Indicates the *application-context-name* parameter of the AARQ and AARE APDUs.

Type	structure of type objid_t (see page 80)
Values:	any valid <i>application-context-name</i> , no default
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_DCS

Conveys the *defined-context-set* which contains the abstract syntax/transfer syntax pairs that were negotiated when the association was established, together with the *presentation-context-identifier* that identifies the pair. It contains both the presentation contexts that were negotiated by the user (using AP_PCDL and AP_PCDRL) and any additional presentation contexts that are required by APLI or the service provider, but which were not negotiated on their behalf.

The value of AP_DCS must be empty (size field of zero) in the states AP_IDLE and AP_WASSOCcnf_ASSOCreq. It must contain the ACSE context negotiated by APLI in the state AP_WASSOCrsp_ASSOCind, and it must contain the fully negotiated defined context set in all other states.

Type	structure of type ap_dcs_t (see page 77)
Values:	any valid <i>defined-context-set</i> , or not present (default)
Readable:	in any state but: AP_UNBOUND
Writable:	never

AP_DPCN

Optional parameter that indicates the *default-context-name* parameter of the CP PPDU. By default, this attribute is not present.

Type	structure of type objid_t (see page 80)
Values:	any valid <i>application-context-name</i> , or not present (default)
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE

AP_DPCR

Indicates the *default-context-result* parameter of the CPR PPDU. Since the user is responsible for encoding and decoding of user-data, it is the user's responsibility to accept or reject any proposed default-presentation-context. The default value `AP_DPCR_NOVAL` indicates that it is not present.

Type	long
Values:	one of: <code>AP_ACCEPT</code> , <code>AP_USER_REJ</code> , or <code>AP_PROV_REJ</code> , <code>AP_DPCR_NOVAL</code> (default)
Readable:	in any state but: <code>AP_UNBOUND</code> and <code>AP_IDLE</code>
Writable:	only in state: <code>AP_WASSOCrsp_ASSOCind</code>

AP_INIT_SYNC_PT

Conveys the desired initial session synchronization point serial number.

Type	unsigned long
Values:	range from <code>AP_MIN_SYNCPT(0)</code> to <code>AP_MAX_SYNCPT(999999)</code>
Readable:	always
Writable:	only in states: <code>AP_UNBOUND</code> , <code>AP_IDLE</code> , and <code>AP_WASSOCrsp_ASSOCind</code>

AP_LCL_PADDR

Holds the value used as the presentation address of the local entity. This attribute is not directly configurable by the user. However, setting the `AP_BIND_PADDR` attribute results in the `AP_LCL_PADDR` attribute being set to the same value.

Type	structure of type <code>ap_paddr_t</code> (see page 78)
Values:	any valid presentation address, no default
Readable:	always
Writable:	never

AP_LIB_AVAIL

Indicates which versions of APLI are available to the service user. For the SunLink implementation of APLI, this attribute is always set to AP_LIBVER1.

Type	unsigned long
Values (bit):	AP_LIBVER1 (default)
Readable:	always
Writable:	never

AP_LIB_SEL

Indicates which version of APLI is currently selected by the service user. For the SunLink implementation of APLI, this attribute is always set to AP_LIBVER1.

Type	unsigned long
Values (bit):	AP_LIBVER1 (default)
Readable:	always
Writable:	only in state: AP_UNBOUND

AP_MODE_AVAIL

Indicates which modes of operation for APLI and the service provider are available to the service user. For the SunLink implementation of APLI, this attribute always has the AP_NORMAL_MODE bit set.

Type	unsigned long
Values (bit):	AP_NORMAL_MODE (default)
Readable:	always
Writable:	never

AP_MODE_SEL

Indicates which modes of operation for APLI and the service provider are available to the service user. For the SunLink implementation of APLI, this attribute always has the AP_NORMAL_MODE bit set.

Type	unsigned long
Values (bit):	AP_NORMAL_MODE (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_MSTATE

Indicates that data is pending. If the APLI instance is awaiting additional data from the user (the last `ap_snd()` call had the AP_MORE bit set), the AP_SNDMORE bit in AP_MSTATE is set. If there is more user data for the current service (the last call to `ap_rcv()` returned with the AP_MORE bit set), AP_RCVMORE bit is set. (It is possible for both bits to be set.)

Type	unsigned long
Values (bit):	AP_SNDMORE, AP_RCVMORE, AP_SNDMORE AP_RCVMORE, or not set (default)
Readable:	always
Writable:	never

AP_PCDL

Lists the *presentation-contexts* proposed by the association-initiator. Since the service user is responsible for all encoding and decoding of user data, the proposed transfer syntaxes for each proposed abstract syntax must be included in the list proposed by the association-initiator user. For the association-responder, the list indicates which presentation contexts are being requested by the association-initiator for use on the association.

The association-initiator may specify the ACSE context in AP_PCDL, although it is not required to do so. If the association-initiator does not supply the ACSE context in AP_PCDL, APLI supplies it automatically for the association.

Regardless of whether or not the association-initiator supplies the ACSE context, it is not provided to the association-responder in AP_PCDL (the responding user may obtain it from AP_DCS if required).

Type	structure of type ap_cdl_t (see page 72)
Values:	any valid <i>presentation-context-definition-list</i> , or not present (default)
Readable:	always
Writable:	only in states: AP_UNBOUND and AP_IDLE

AP_PCDRL

Indicates whether the presentation contexts proposed in the presentation context definition list have been accepted or rejected. The association-responder uses this attribute to indicate which of the proposed presentation contexts are acceptable before issuing the A_ASSOC_RSP primitive. For the association-initiator, this attribute indicates the remote user's response to the proposed presentation contexts received in the A_ASSOC_CNF primitive.

Each entry in AP_PCDRL corresponds one-to-one with an entry in AP_PCDL, which, in the case of the responder, never contains the proposed ACSE context. The association-responder has an opportunity to accept or reject each of the proposed contexts that are indicated in AP_PCDL. The ACSE context is automatically accepted by APLI and appears in the value of AP_DCS.

Type	structure of type ap_cdrl_t (see page 73)
Values:	any valid <i>presentation-context-definition-response-list</i> or not present (default)
Readable:	in any state but: AP_UNBOUND and AP_IDLE
Writable:	only in states: AP_WASSOCrsp_ASSOCind and AP_WASSOCcnf_ASSOCreq

AP_PFU_AVAIL

Indicates which optional presentation functional units are currently available. For the SunLink implementation of APLI this attribute always has a `NULL` value.

Type	unsigned long
Values (bit):	<code>NULL</code>
Readable:	always
Writable:	never

AP_PFU_SEL

Indicates which optional presentation layer functional units have been requested for use over the current association. For the SunLink implementation of APLI this attribute always has a `NULL` value.

Type	unsigned long
Values (bit):	<code>NULL</code>
Readable:	always
Writable:	never

AP_PRES_AVAIL

The `AP_PRES_AVAIL` attribute indicates which versions of the presentation layer protocol are currently available. For the SunLink implementation of APLI this attribute always has `AP_PRESVER1` bit set.

Type	unsigned long
Values (bit):	<code>AP_PRESVER1</code> (default)
Readable:	always
Writable:	never

AP_PRES_SEL

The AP_PRES_SEL attribute indicates which version of the presentation layer protocol has been selected for use with the current association. For the SunLink implementation of APLI this attribute always has AP_PRESVER1 bit set.

Type	unsigned long
Values (bit):	AP_PRESVER1 (default)
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_REM_PADDR

The AP_REM_PADDR attribute holds the value used as the presentation address of the remote entity. If the local entity is the initiator of an association, this value is the called presentation address. If the local entity is the association-responder, this value is the calling presentation address.

Type	structure of type ap_paddr_t (see page 78)
Values:	any valid presentation address, no default
Readable:	always
Writable:	only in states : AP_UNBOUND and AP_IDLE

AP_ROLE_ALLOWED

Indicates the role played by the APLI instance during association negotiation.

If the AP_INITIATOR bit of AP_ROLE_ALLOWED is set, the APLI instance is able to issue association requests (A_ASSOC_REQ) and is prevented from accepting association indications (A_ASSOC_IND).

If the RESPONDER bit of AP_ROLE_ALLOWED is set, the APLI instance is able to accept association indications (A_ASSOC_IND) and is prevented from issuing association requests (A_ASSOC_REQ).

If both the `AP_INITIATOR` bit and the `AP_RESPONDER` bit are set, the APLI instance can both issue association requests (`A_ASSOC_REQ`) and accept association indications (`A_ASSOC_IND`). This is the default value assigned to the attribute. The role played by the APLI instance in the current association is indicated by the environment attribute `AP_ROLE_CURRENT` which is set to `AP_INITIATOR` when an association request (`A_ASSOC_REQ`) is issued and set to `AP_RESPONDER` when an association indication (`A_ASSOC_IND`) is received.

Note that an APLI instance may still accept an association indication (`A_ASSOC_IND`) after the value of `AP_ROLE_ALLOWED` has been set to `AP_INITIATOR`, provided that the APLI instance is in an idle state and the association indication was queued prior to setting `AP_ROLE_ALLOWED`. This condition can be avoided by ensuring that `AP_ROLE_ALLOWED` is set before the APLI instance is bound to a presentation address.

Type	unsigned long
Values (bit):	<code>AP_INITIATOR</code> <code>AP_RESPONDER</code> <code>AP_INITIATOR</code> <code>AP_RESPONDER</code> (default)
Readable:	always
Writable:	always

AP_ROLE_CURRENT

The `AP_ROLE_CURRENT` attribute indicates the role of the local user in the current association. The attribute is set to `AP_INITIATOR` as soon as an `A_ASSOC_REQ` primitive is sent and remains unchanged until the association is rejected or subsequently terminated. Similarly, the attribute is set to `AP_RESPONDER` upon receipt of an `A_ASSOC_IND` and left unchanged until the association is terminated.

Type	unsigned long
Values (bit):	<code>AP_INITIATOR</code> , <code>AP_RESPONDER</code> <code>AP_INITIATOR</code> <code>AP_RESPONDER</code> (default)
Readable:	in any states but: <code>AP_UNBOUND</code> and <code>AP_IDLE</code>
Writable:	never

AP_RSP_AEID

The AP_RSP_AEID is the responding *AE-invocation-identifier* of the AARE APDU.

Type	long
Values (bit):	any valid AEID, AP_RSP_AEID_NOVAL (default)
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_RSP_AEQ

The AP_RSP_AEQ is the responding *AE-qualifier* of the AARE APDU. By default, this attribute is not present as indicated by setting the *size* of **any_t** to -1.

Type	structure of type any_t (see page 79)
Values:	any valid AEQ, not present by default
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_RSP_APID

The AP_RSP_APID is the responding *AP-invocation-identifier* of the AARE APDU.

Type	long
Values (bit):	any valid AEID, AP_RSP_APID_NOVAL (default)
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_RSP_APT

The AP_RSP_APT is the responding *AP-title* parameter of the AARE APDU.

Type	structure of type any_t (see page 79)
Values:	any valid APT, not present by default
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_SESS_AVAIL

The AP_SESS_AVAIL attribute indicates which versions of the session layer protocol are currently available.

Type	unsigned long
Values (bit):	OR'ed bits AP_SESSVER1 AP_SESSVER2
Readable:	always
Writable:	never

AP_SESS_SEL

The AP_SESS_SEL attribute indicates which version of the session layer protocol has been selected for use with the current association.

Type	unsigned long
Values (bit):	OR'ed bits AP_SESSVER1 AP_SESSVER2 (default)
Readable:	always
Writable:	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_SFU_AVAIL

The AP_SFU_AVAIL attribute indicates which session layer functional units are currently available. The default value is all bits set. The value of this attribute may affect the value of AP_TOKENS_AVAIL.

Type	unsigned long
Values (bit):	OR'ed bits: AP_SESS_DUPLEX AP_SESS_HALFDUPLEX AP_SESS_XDATA AP_SESS_TDATA AP_SESS_MINORSYNC AP_SESS_MAJORSYNC AP_SESS_RESYNC AP_SESS_EXCEPT AP_SESS_ACTMGMT
Readable:	always
Writable	never

AP_SFU_SEL

The AP_SFU_SEL attribute indicates which session layer functional units have been requested for use over the current association.

Type	unsigned long
Values (bit):	OR'ed bits: AP_SESS_DUPLEX AP_SESS_HALFDUPLEX AP_SESS_XDATA AP_SESS_TDATA AP_SESS_MINORSYNC AP_SESS_MAJORSYNC AP_SESS_RESYNC AP_SESS_EXCEPT AP_SESS_ACTMGMT
Readable:	always
Writable	only in states: AP_UNBOUND, AP_IDLE, and AP_WASSOCrsp_ASSOCind

AP_STATE

The `AP_STATE` attribute is used to convey state information about the APLI interface. It is used to determine which primitives are legal, which attributes can be read/written, etc.

Type	unsigned long
Values (bit):	OR'ed bits: <code>AP_UNBOUND</code> (default) <code>AP_IDLE</code> <code>AP_DATA_XFER</code> <code>AP_WASSOCrsp_ASSOCind</code> <code>AP_WASSOCrsp_ASSOCind</code>
Readable:	always
Writable	never

AP_STREAM_FLAGS

Controls the characteristics of the APLI instance. When an APLI instance is opened with the `AP_NDELAY` bit set, it operates in non-blocking execution mode.

Type	long
Values (bit):	<code>AP_NDELAY</code> , <code>NULL</code> (default)
Readable:	always
Writable:	always

AP_TOKENS_AVAIL

A token is an attribute of an association that is dynamically assigned to one user at a time. The user that possesses a token has exclusive rights to initiate the service that token represents. The `AP_TOKENS_AVAIL` attribute indicates which tokens are available for assignment on this association. The value of this attribute is dependent on `AP_SFU_SEL`.

Type	unsigned long
Values (bit):	NULL (default) AP_DATA_TOK AP_SYNCMAJOR_TOK AP_SYNCMINOR_TOK AP_RELEASE_TOK
Readable:	always
Writable	never

AP_TOKENS_OWNED

The `AP_TOKENS_OWNED` attribute indicates which available tokens (see `AP_TOKENS_AVAIL`) are currently assigned to the user. The user has exclusive rights to initiate the service represented by each of the tokens owned. The value of this attribute is affected also by `AP_INIT_TOKENS`.

Type	unsigned long
Values (bit):	NULL (default) AP_DATA_TOK AP_SYNCMAJOR_TOK AP_SYNCMINOR_TOK AP_RELEASE_TOK
Readable:	in any states but: AP_UNBOUND, AP_IDLE, AP_WASSOCrsp_ASSOCind, and AP_WASSOCrsp_ASSOCind
Writable	never

Environment Data Types

The following data types are defined in the header file `<ap_lib.h>` and are used to convey values associated with environment attributes.

`ap_cdl_t`

Name

`ap_cdl_t`—used to convey context definition lists.

Synopsis

```
typedef struct {
    int    size_ap_cdl;          /* size of array */
    struct seqof_ap_cdl {
        long    pci;
        objid_t *a_sytx;        /* ptrs to abstract syntaxes */
        int     size_t_sytx;    /* number of transfer syntaxes */
        objid_t **m_t_sytx;     /* ptrs to transfer syntaxes */
    } *m_ap_cdl;                /* array */
} ap_cdl_t;
```

Description

The context definition list comprises a series of elements in the array *seqof_ap_cdl*. The number of elements in this array is given as *size_ap_cdl*.

The presentation context identifier is represented by *seqof_ap_cdl[i].pci*. *seqof_ap_cdl[i].a_sytx* is a pointer to an **objid_t** and is used to convey the abstract syntax name associated with the presentation context identifier. *seqof_ap_cdl[i].m_t_sytx* is a pointer to an array of type **objid_t ***, which includes the transfer syntaxes that the association-initiator is capable of supporting for the named abstract syntax. *seqof_ap_cdl[i].size_t_sytx* is the number of elements in the *seqof_ap_cdl[i].m_t_sytx* array.

The context definition list is an optional parameter of the CP PPDU. Setting *size* to `-1` indicates that this parameter is not present. The presentation context definition list may also be specified as absent by invoking `ap_set_env()` with `AP_PCDL` as the *attr* argument and a `NULL` pointer as the *val* argument.

ap_cdrl_t

Name

ap_cdrl_t—used to convey context definition result lists.

Synopsis

```
/* AP_PCDRL.res */
#define ACCPT                (0)
#define USER_REJ            (1)
#define PROV_REJ            (2)
/* AP_PCDRL.prov_rsn */
#define RSN_NSPEC            (0)
#define A_SYTX_NSUP         (1)
#define PROP_T_SYTX_NSUP    (2)
#define LCL_LMT_DCS_EXCEEDED (3)

typedef struct {
    int    size_ap_cdrl;          /* size of array */
    struct seqof_ap_cdrl {
        long    res;             /* result of negotiation */
        objid_t *t_sytx;         /* negotiated transfer syntax */
        long    prov_rsn;        /* reason for rejection */
    } *m_ap_cdrl;               /* array */
} ap_cdrl_t;
```

Description

The context definition result list comprises a series of elements in the array, *seqof_ap_cdrl*. The number of elements in this array is given as *size_ap_cdrl*.

There is a one-to-one correspondence between the elements of a context definition list and those in the related context definition result list. For each entry in the presentation context definition list, the *seqof_ap_cdrl[i].res* field of the corresponding element in the context definition result list must be set by the association-responder to one of ACCPT, USER_REJ, or PROV_REJ. If *seqof_ap_cdrl[i].res* is set to ACCPT, then *seqof_ap_cdrl[i].t_sytx* indicates the transfer syntax selected by the association-responder. If *seqof_ap_cdrl[i].res* is set to either USER_REJ or PROV_REJ, *seqof_ap_cdrl[i].prov_rsn* is set to the reason for the rejection of the abstract syntax.

The possible values for *m_ap_cdr[i].prov_rsn* are:

RSN_NSPEC (reason not specified)

A_SYTX_NSUP (abstract syntax not supported)

PROP_T_SYTX_NSUP (proposed transfer syntaxes not supported)

LCL_LMT_DCS_EXCEEDED (local limit on DCS exceeded)

The context definition result list is an optional parameter of the CPA and CPR PPDUs. Setting the value of *size* to -1 indicates that this parameter is not present. The presentation context definition result list may also be specified as absent by invoking `ap_set_env()` with `AP_PCDRL` as the *attr* argument and a NULL pointer as the *val* argument.

ap_conn_id_t

Name

ap_conn_id_t —used to convey session connection identifiers.

Synopsis

```
/* AP_OCTET */
typedef struct {
    long    length;                /* number of octets */
    unsigned char *data; /* octets */
} ap_octet_string_t;

typedef struct {
    ap_octet_string_t *user_ref;    /* user ref. */
    ap_octet_string_t *comm_ref;    /* common ref. */
    ap_octet_string_t *addtl_ref;   /* additional ref. */
} ap_conn_id_t;
```

Description

Each member of the **ap_conn_id_t** structure a pointer to a structure of type **ap_octet_string_t**. The *user_ref* member must be ≤ 64 octets. The *comm_ref* member must be ≤ 64 octets. The *addtl_ref* member must be ≤ 4 octets. The absence of a particular member of a connection identifier may be indicated either by setting the corresponding field to NULL, or by specifying a 0 length **ap_octet_string_t**.

`ap_dcn_t`

Name

ap_dcn_t—used to convey default context names.

Synopsis

```
typedef struct {
    ap_objid_t  *a_sytx; /* abstract syntax name */
    ap_objid_t  *t_sytx; /* transfer syntax name */
} ap_dcn_t;
```

Description

Both *a_sytx* and *t_sytx* are pointers to objects of type **ap_objid_t**: *a_sytx* points to the abstract syntax for the default presentation context, while *t_sytx* points to the transfer syntax for the default presentation context.

The default context name is an optional parameter of the CP PPDU. Setting both *a_sytx* and *t_sytx* to `NULL` indicates that this parameter is not present. The default context name may also be specified to be absent by invoking `ap_set_env()` with `AP_DPCN` as the *attr* argument and a `NULL` pointer as the *val* argument.

ap_dcs_t

Name

ap_dcs_t—used to convey the defined context set.

Synopsis

```
typedef struct {
    int size;                /* number of elements */
    struct dcs_elt {
        int pci;            /* presentation context ID */
        objid_t *a_sytx;    /* abstract syntax */
        objid_t *t_sytx;    /* transfer syntax */
    } *dcs;                 /* array */
} ap_dcs_t;
```

Description

The *size* field of the **ap_dcs_t** structure indicates the number of elements in the defined context set. Each element consists of a presentation context identifier (*dcs_elt[i].pci*), an abstract syntax name (*dcs_elt[i].a_sytx*), and a transfer syntax name (*dcs_elt[i].t_sytx*).

ap_paddr_t

Name

ap_paddr_t—used to convey presentation addresses.

Synopsis

```
typedef struct {
    ap_octet_string_t *p_selector; /* Presentation selector */
    ap_octet_string_t *s_selector; /* Session selector */
    ap_octet_string_t *t_selector; /* Transport selector */
    int n_nsaps; /* Number of network addrs */
    ap_octet_string_t *nsaps; /* Network address */
} ap_paddr_t;
```

Description

The pointers *p_selector*, *s_selector*, and *t_selector* point to structures of type **ap_octet_string_t** that contain the presentation, session, and transport selectors respectively. The pointer *nsaps* points to an array of structures of type **ap_octet_string_t** that contain the network address. The *n_nsaps* element is used to specify how many network address components are in the array. For the SunLink implementation of APLI, this must always be set to 1.

The structure **ap_octet_string_t** is defined as:

```
typedef struct {
    long length; /* number of octets */
    unsigned char*data; /* octets */
} ap_octet_string_t;
```

The *length* field of this structure indicates the number of octets in the octet string pointed to by *data*. To specify a null selector value, the length field of the **ap_octet_string_t** structure is set to 0. Presentation addresses are restricted as follows:

- Session selectors cannot exceed 16 octets.
- Transport selectors cannot exceed 32 octets.
- Network addresses cannot exceed 20 octets.

any_t

Name

any_t—used to convey ASN.1 objects of type ANY.

Synopsis

```
typedef struct {  
    int    size;  
    char  *udata;  
} any_t
```

Description

Structures of type **any_t** are used to convey ASN.1 objects of type ANY (for example, the attribute AP_CLD_AEQ).

The pointer *udata* points to a buffer that contains user data to be conveyed and the **integer** *size* specifies the length of this buffer. For optional PDU parameters of type ANY, setting *size* to -1 indicates that the parameter is not present.

An optional PDU parameter can also be specified as not present by calling the function `ap_set_env()` with a NULL pointer as the *val* argument.

objid_t

Name

objid_t —used to convey ASN.1 objects of type OBJECT IDENTIFIER.

Synopsis

```
#define AP_MAXOBJBUF 12

typedef struct {
    long length; /* Number of value octets in object ID encoding */
    union {
        unsigned char *long_buf;
        unsigned char short_buf[AP_MAXOBJBUF];
    } b;
} objid_t;
```

Description

The **objid_t** structure is used to convey OBJECT IDENTIFIER values. OBJECT IDENTIFIER values are stored as the contents octets of their encoded form (without the tag or length octets).

If the number of contents octets is greater than MAXOBJBUF, the contents octets are stored beginning at the memory location pointed to by **long_buf**. Otherwise, the contents octets are stored in the **short_buf** array. In both cases, **length** gives the number of contents octets in the OBJECT IDENTIFIER encoding. The absence of an OBJECT IDENTIFIER parameter is indicated by setting the **length** field to 0.

Environment Read/Write Table

Table 4-2 summarizes the valid values and read/write states for the environment attributes described in this chapter.

Table 4-2 Attribute Read/Write Table

Attribute	Type/Values	Readable	Writable
AP_ACSE_AVAIL	unsigned long bit values: AP_ACSEVER1 default: AP_ACSEVER1	always	never
AP_ACSE_SEL	unsigned long bit values: AP_ACSEVER1 default: AP_ACSEVER1	always	only in states: AP_UNBOUND AP_IDLE
AP_BIND_PADDR	ap_paddr_t default: none	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_CLD_AEID	long default: AP_CLD_AEID_NOVAL	always	only in states: AP_UNBOUND AP_IDLE
AP_CLD_AEQ	any_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLD_APID	long default: AP_CLD_APID_NOVAL	always	only in states: AP_UNBOUND AP_IDLE
AP_CLD_APT	any_t default: not present	always	only in states: AP_UNBOUND AP_IDLE

Table 4-2 Attribute Read/Write Table (Continued)

Attribute	Type/Values	Readable	Writable
AP_CLD_CONN_ID	ap_cld_conn_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_CLG_AEID	long default: AP_CLG_AEID_NOVAL	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_AEQ	any_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_APID	long default: AP_CLG_APID_NOVAL	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_APT	any_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_CONN_ID	ap_clg_conn_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CNTX_NAME	objid_t default: none	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_DCS	ap_dcs_t default: not present	in any states but: AP_UNBOUND	never
AP_DPCN	objid_t default: not present	always	only in states: AP_UNBOUND AP_IDLE

Table 4-2 Attribute Read/Write Table (Continued)

Attribute	Type/Values	Readable	Writable
AP_DPCR	long one of: AP_DPCR_NOVAL AP_ACCEPT AP_USER_REJ AP_PROV_REJ default: AP_DPCR_NOVAL	in any state but: AP_UNBOUND AP_IDLE	only in state(s): AP_WASSOCrsp_ASSOCind
AP_INIT_SYNC_PT	unsigned long range from AP_MIN_SYNCP(0) to AP_MAX_SYNCP(999999) default: AP_MIN_SYNCP	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_LCL_PADDR	ap_paddr_t	always	never
AP_LIB_AVAIL	unsigned long bit values: AP_LIBVER1 default (or'ed bits): AP_LIBVER1	always	never
AP_LIB_SEL	unsigned long bit values: AP_LIBVER1 default: AP_LIBVER1	always	only in state: AP_UNBOUND
AP_MODE_AVAIL	unsigned long bit values: AP_NORMAL_MODE default: AP_NORMAL_MODE	always	never

Table 4-2 Attribute Read/Write Table (Continued)

Attribute	Type/Values	Readable	Writable
AP_MODE_SEL	unsigned long bit values: AP_NORMAL_MODE default: AP_NORMAL_MODE	always	only in states: AP_UNBOUND AP_IDLE
AP_MSTATE	unsigned long bit values: AP_SNDMORE AP_RCVMORE default (or'ed bits): NULL	always	never
AP_PCDL	ap_cdl_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_PCDRL	ap_cdr1_t default: not present	in any state but: AP_UNBOUND AP_IDLE	only in states: AP_WASSOCrsp_ASSOCind AP_WASSOCrsp_ASSOCreq
AP_PFU_AVAIL	unsigned long bit values: NULL default (or'ed bits): NULL	always	never
AP_PFU_SEL	unsigned long bit values: NULL default: NULL	always	never

Table 4-2 Attribute Read/Write Table (Continued)

Attribute	Type/Values	Readable	Writable
AP_PRES_AVAIL	unsigned long bit values: AP_PRESVER1 default (or'd bits): AP_PRESVER1	always	never
AP_PRES_SEL	unsigned long bit values: AP_PRESVER1 default: AP_PRESVER1	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_REM_PADDR	ap_paddr_t default: none	always	only in states: AP_UNBOUND AP_IDLE
AP_ROLE_ALLOWED	unsigned long bit values: AP_INITIATOR AP_RESPONDER default: AP_INITIATOR AP_RESPONDER	always	always
AP_ROLE_CURRENT	unsigned long bit values: AP_INITIATOR AP_RESPONDER default: both set	in any states but: AP_UNBOUND AP_IDLE	never
AP_RSP_AEID	long default: AP_RSP_AEID_NOVAL	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind

Table 4-2 Attribute Read/Write Table (Continued)

Attribute	Type/Values	Readable	Writable
AP_RSP_AEQ	any_t default: not present	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_RSP_APID	long default: AP_RSP_APID_NOVAL	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_RSP_APT	any_t default: not present	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_SESS_AVAIL	unsigned long bit values: AP_SESSVER1 AP_SESSVER2 default: AP_SESSVER1 AP_SESSVER2	always	never
AP_SESS_SEL	unsigned long bit values: AP_SESSVER1 AP_SESSVER2 default: AP_SESSVER2	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind

Table 4-2 Attribute Read/Write Table (Continued)

Attribute	Type/Values	Readable	Writable
AP_SFU_AVAIL	unsigned long or'ed bits begin with LSB: AP_SESS_DUPLEX AP_SESS_HALFDUPLEX AP_SESS_XDATA AP_SESS_TDATA AP_SESS_MINORSYNC AP_SESS_MAJORSYNC AP_SESS_RESYNC AP_SESS_EXCEPT AP_SESS_ACTMGMT default: all set	always	never
AP_SFU_SEL	unsigned long or'ed bits: AP_SESS_DUPLEX AP_SESS_HALFDUPLEX AP_SESS_XDATA AP_SESS_TDATA AP_SESS_MINORSYNC AP_SESS_MAJORSYNC AP_SESS_RESYNC AP_SESS_EXCEPT AP_SESS_ACTMGMT default: all set	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind

≡ 4

Table 4-2 Attribute Read/Write Table (Continued)

Attribute	Type/Values	Readable	Writable
AP_STATE	unsigned long one of: AP_UNBOUND AP_IDLE AP_DATA_XFER AP_WASSOCrsp_ASSOCind AP_WASSOCcnf_ASSOCreq default: AP_UNBOUND	always	never
AP_STREAM_FLAGS	long bit values: AP_NDELAY default: NULL	always	always
AP_TOKENS_AVAIL	unsigned long bit values: AP_DATA_TOK AP_SYNCMINOR_TOK AP_MAJACT_TOK AP_RELEASE_TOK default (or'ed bits): NULL	always	never
AP_TOKENS_OWNED	unsigned long bit values: AP_DATA_TOK AP_SYNCMINOR_TOK AP_MAJACT_TOK AP_RELEASE_TOK default (or'ed bits): NULL	in any states but: AP_UNBOUND AP_IDLE AP_WASSOCcnf_ASSOCreq AP_WASSOCrsp_ASSOCind	never

<i>A_ABORT_REQ—request abnormal release</i>	<i>page 93</i>
<i>A_ABORT_IND—indicate abnormal release</i>	<i>page 95</i>
<i>A_ASSOC_REQ—request association</i>	<i>page 97</i>
<i>A_ASSOC_IND—indicate association request</i>	<i>page 100</i>
<i>A_ASSOC_RSP—respond to association request</i>	<i>page 102</i>
<i>A_ASSOC_CNF—confirm association request</i>	<i>page 106</i>
<i>A_PABORT_REQ—request abnormal provider release</i>	<i>page 110</i>
<i>A_PABORT_IND—indicate abnormal provider release</i>	<i>page 114</i>
<i>A_RELEASE_REQ—request normal release</i>	<i>page 118</i>
<i>A_RELEASE_IND—indicate normal release request</i>	<i>page 120</i>
<i>A_RELEASE_RSP—respond to normal release request</i>	<i>page 122</i>
<i>A_RELEASE_CNF—confirm normal release request</i>	<i>page 124</i>
<i>P_DATA_REQ—send user data</i>	<i>page 126</i>
<i>P_DATA_IND—indicate receipt of user data</i>	<i>page 128</i>
<i>P_TOKENGIVE_REQ—give session token</i>	<i>page 130</i>
<i>P_TOKENGIVE_IND—indicate receipt of token</i>	<i>page 132</i>
<i>P_TOKENPLEASE_REQ—request token</i>	<i>page 134</i>
<i>P_TOKENPLEASE_IND—indicate request for token</i>	<i>page 136</i>

Each APLI primitive corresponds to one of the services provided by the underlying OSI protocol.

APLI service primitives are exchanged using calls to the APLI functions `ap_snd()` and `ap_rcv()` as described in Chapter 3, “APLI Functions” and associated protocol information is transmitted in the union of structures `ap_cdata_t`.

Table 5-1 summarizes the relationships between each of the primitives and the states and attributes associated with them. The following information is provided:

valid in states

Lists the states (returned by the attribute `AP_STATE`) in which each primitive can be sent or received.

next state

Lists the next state (returned by the attribute `AP_STATE`) entered as a result of sending or receiving each primitive.

may change

Lists the attributes that may change as a result of sending or receiving each primitive.

Table 5-1 Primitive/State/Attribute Relationships

Primitive	valid in states	next state	may change
A_ABORT_REQ	all except: AP_UNBOUND AP_IDLE	AP_IDLE	AP_STATE
A_ABORT_IND	all except: AP_UNBOUND AP_IDLE	AP_IDLE	AP_STATE
A_PABORT_REQ	all except: AP_UNBOUND AP_IDLE	AP_IDLE	AP_STATE
A_PABORT_IND	all except: AP_UNBOUND	AP_IDLE	AP_STATE
A_ASSOC_REQ	AP_IDLE	AP_WASSOCcnf_ASSOCreq	AP_ROLE_CURRENT AP_STATE

Table 5-1 Primitive/State/Attribute Relationships

Primitive	valid in states	next state	may change
A_ASSOC_IND	AP_IDLE	AP_WASSOCrsp_ASSOCind	AP_ACSE_SEL AP_CLD_AEID & AP_CLG_AEID AP_CLD_APID & AP_CLG_APID AP_CLD_AEQ & AP_CLG_AEQ AP_CLD_APT & AP_CLG_APT AP_CNTX_NAME AP_DCS AP_DPCN AP_INIT_SYNC_PT AP_INIT_TOKENS AP_LCL_PADDR AP_MODE_SEL AP_PCDL AP_PFU_SEL AP_PRES_SEL AP_QOS AP_REM_PADDR AP_ROLE_CURRENT AP_SESS_SEL AP_SFU_SEL AP_STATE AP_TOKENS_AVAIL AP_TOKENS_OWNED
A_ASSOC_RSP	AP_WASSOCrsp_ASSOCind	AP_IDLE or AP_DATA_XFER	AP_DCS AP_STATE AP_TOKENS_OWNED

Table 5-1 Primitive/State/Attribute Relationships

Primitive	valid in states	next state	may change
A_ASSOC_CNF	AP_WASSOCcnf_ASSOCreq	AP_IDLE or AP_DATA_XFER	AP_ACSE_SEL AP_AFU_SEL AP_CNTX_NAME AP_DCS AP_INIT_SYNC_PT AP_INIT_TOKENS AP_PFU_SEL AP_PCDRL AP_PRES_SEL AP_QOS AP_REM_PADDR AP_SESS_SEL AP_SFU_SEL AP_STATE AP_TOKENS_AVAIL AP_TOKENS_OWNED
A_RELEASE_REQ	AP_DATA_XFER	AP_WRELcnf_RELreq	AP_STATE
A_RELEASE_IND	AP_DATA_XFER AP_WRELcnf_RELreq	AP_IDLE or AP_DATA_XFER	AP_STATE
A_RELEASE_RSP	AP_WRELrsp_RELind AP_WRELrsp_RELind_init	AP_IDLE or AP_DATA_XFER AP_WRELcnf_RELreq	AP_STATE
A_RELEASE_CNF	AP_WRELcnf_RELreq AP_WRELcnf_RELreq_rsp	AP_IDLE or AP_DATA_XFER AP_WRELrsp_RELind	AP_STATE
P_DATA_REQ	AP_DATA_XFER AP_WRELrsp_RELind	no state change	none
P_DATA_IND	AP_DATA_XFER AP_WRELcnf_RELreq	no state change	none
P_TOKENGIVE_REQ	AP_DATA_XFER	no state change	AP_STATE AP_TOKENS_OWNED
P_TOKENGIVE_IND	AP_DATA_XFER	no state change	AP_STATE AP_TOKENS_OWNED
P_TOKENPLEASE_REQ	AP_DATA_XFER AP_WRELrsp_RELind	no state change	none
P_TOKENPLEASE_IND	AP_DATA_XFER AP_WRELcnf_RELreq	no state change	none

A_ABORT_REQ—*request abnormal release*

Name

A_ABORT_REQ—request abnormal release of an association.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to A_ABORT_REQ */
    ap_cdata_t   *cdata, /* ptr to type a_abort_req_cd_t */
    struct osi_buf *ubuf,
    int          flags
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function `ap_snd()`.

Description

The A_ABORT_REQ primitive is used in conjunction with `ap_snd()` and the APLI environment to request the abnormal release of an association.

To issue an abort request, the symbolic constant *sptype* must be set to A_ABORT_REQ and the pointer *cdata* must point to a structure of type **a_abort_req_cd_t**, which is defined as follows:

```
typedef struct {
    long    udata_length; /* length of user data */
} a_abort_req_cd_t;
```

If the primitive is issued using multiple calls to `ap_snd()` with the AP_MORE flag set, the argument `cdata→udata_length` must be set to the total number of octets of user information to be sent in the **osi_buf** structure pointed to by *ubuf*. This argument is ignored if the primitive is issued as a single `ap_snd()`.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_BADENV`, `AP_BADLSTATE`, `BAD_PRIM`,
`BAD_UBUF`, `BAD_HANGUP`, `AP_NOENV`

A_ABORT_IND—*indicate abnormal release*

Name

A_ABORT_IND—used to indicate an abort request.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns A_ABORT_IND */
    ap_cdata_t   *cdata,  /* ptr to type a_abort_ind_cd_t */
    struct osi_buf *ubuf,
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The A_ABORT_IND primitive is used in conjunction with ap_rcv() and the APLI environment to indicate the abnormal release of an association, or the abnormal termination of either the A-ASSOCIATE or the A-RELEASE service.

To receive an abort indication, the pointer *sptype* must point to an **unsigned long** and the pointer *cdata* must point to a structure of type **a_abort_ind_cd_t**, which is defined as follows:

```
typedef struct {
    long   src;           /* source of abort request */
    long   udata_length  /* length of user data */
} a_abort_ind_cd_t;
```

On return, the **unsigned long** pointed to by *sptype* is set to `A_ABORT_IND` and any additional protocol information is returned in the structure pointed to by *cdata*.

The value of *cdata*→*src* is set to indicate the source of the abort request. The possible values for *cdata*→*src* when the APLI is operating in *normal* mode are:

`AP_ACSE_USER` (abort requested by ACSE user)

`AP_ACSE_PROV` (abort requested by ACSE provider)

The value of *cdata*→*udata_length* is set to indicate the total number of octets of user data received with the primitive in the **osi_buf** structure pointed to by *ubuf*.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_PDUREJ`, `AP_BADLSTATE`, `AP_NOBUF`

A_ASSOC_REQ—*request association*

Name

A_ASSOC_REQ—used to initiate an association.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to A_ASSOC_REQ */
    ap_cdata_t   *cdata, /* ptr to type a_assoc_req_cd_t */
    struct osi_buf *ubuf,
    int          flags
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function ap_snd().

Description

The A_ASSOC_REQ primitive is used in conjunction with ap_snd() and the APLI environment to request and initiate an association.

Before issuing an association request the association must be configured by setting the attributes AP_BIND_PADDR, AP_CNTX_NAME, AP_LIB_SEL, and AP_REM_PADDR.

To issue an association request, the symbolic constant *sptype* must be set to A_ASSOC_REQ and the pointer *cdata* must point to a structure of type **a_assoc_req_cd_t**, which is defined as follows:

```
typedef struct {
    ulong tokens;          /* initial token assignment */
    long  udata_length;    /* length of user data */
} a_assoc_req_cd_t;
```

If the primitive is issued using multiple calls to `ap_snd()` with the `AP_MORE` flag set, the argument `cdata→udata_length` must be set to the total number of octets of user information to be sent in the `osi_buf` structure pointed to by `ubuf`. This argument is ignored if the primitive is issued as a single `ap_snd()`.

The argument `cdata→tokens` is used to specify the initial token assignment, which is defined by OR'ing the values for each available token selected from the following table:

Token	Allowed Values
data	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE
synchronize_minor	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE
major/activity	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE
release	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE

Assignments for tokens that are unavailable are ignored. An invalid combination of tokens causes `ap_snd()` to fail.

Data sent using the `A_ASSOC_REQ` primitive must be BER encoded. This form of encoding is defined for ACSE in X.227, and uses the tag-length-parameter format. The first data octet must be 0xBE. This corresponds to a tag value of 30. The second octet is the length. It can be either the actual length, or the indefinite length value of 0x80. The actual data follows. This is passed transparently, and there are no restrictions to its content. Note that if you set a definite length value, it will be changed internally to an indefinite length value. However, this has no effect on the contents or validity of the data.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_BADENV`, `AP_BADLSTATE`, `BAD_PRIM`,
`BAD_UBUF`, `BAD_HANGUP`, `AP_NOENV`, `AP_BADROLE`, `AP_BADCD_TOKENS`

A_ASSOC_IND—*indicate association request*

Name

A_ASSOC_IND—used to indicate a request to establish an association.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns A_ASSOC_IND */
    ap_cdata_t   *cdata,  /* ptr to type a_assoc_ind_cd_t */
    struct osi_buf *ubuf,
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The A_ASSOC_IND primitive is used in conjunction with ap_rcv() and the APLI environment to indicate a request to establish an association between two application entities.

Before receiving an association indication the association must be configured by setting the attributes AP_BIND_PADDR, and AP_LIB_SEL.

To receive an association indication, the pointer *sptype* must point to an **unsigned long** and the pointer *cdata* must point to a structure of type **a_assoc_ind_cd_t**, which is defined as follows:

```
typedef struct {
    ulong tokens;           /* initial token assignment */
    long  udata_length;    /* length of user data */
} a_assoc_ind_cd_t;
```

On return, the **unsigned long** pointed to by *sptype* is set to `A_ASSOC_IND` and any additional protocol information is returned in the structure pointed to by *cdata*.

The value of *cdata*→*udata_length* is set to indicate the total number of octets of user data received with the primitive in the **osi_buf** structure pointed to by *ubuf*.

The value of *cdata*→*tokens* is set to convey the initial token setting proposed by the initiator, which is defined by OR'ing the values for each available token selected from the following table:

Token	Allowed Values
data	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE
synchronize_minor	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE
major/activity	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE
release	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_PDUREJ`, `AP_BADLSTATE`, `AP_NOBUF`

A_ASSOC_RSP—*respond to association request*

Name

A_ASSOC_RSP—used to respond to an association request indication.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to A_ASSOC_RSP */
    ap_cdata_t    *cdata, /* ptr to type a_assoc_rsp_cd_t */
    struct osi_buf *ubuf,
    int          flags
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function ap_snd().

Description

The A_ASSOC_RSP primitive is used in conjunction with ap_snd() and the APLI environment to respond to an association establishment request.

To issue an association response, the symbolic constant *sptype* must be set to A_ASSOC_RSP and the pointer *cdata* must point to a structure of type **a_assoc_rsp_cd_t**, which is defined as follows:

```
typedef struct {
    long   res;           /* result of negotiation */
    long   diag;          /* reason (if rejected) */
    ulong  tokens;        /* initial token assignment */
    long   udata_length;  /* length of user data */
} a_assoc_rsp_cd_t;
```


If the primitive is issued using multiple calls to `ap_snd()` with the `AP_MORE` flag set, the argument `cdata→udata_length` must be set to the total number of octets of user information to be sent in the `osi_buf` structure pointed to by `ubuf`. This argument is ignored if the primitive is issued as a single `ap_snd()`.

The argument, `cdata→tokens` is used to specify the assignment for tokens which were identified as “responder’s choice” by the initiator that issued the `A_ASSOC_REQ`. The assignment is defined by OR’ing the values for each available token selected from the following table. Assignments for tokens that are unavailable are ignored. An invalid assignment for an available token will cause `ap_snd()` to fail.

Token	Allowed Values
data	one of: <code>AP_DATA_TOK_REQ</code> , <code>AP_DATA_TOK_ACPT</code>
synchronize_minor	one of: <code>AP_DATA_TOK_REQ</code> , <code>AP_DATA_TOK_ACPT</code>
major/activity	one of: <code>AP_DATA_TOK_REQ</code> , <code>AP_DATA_TOK_ACPT</code>
release	one of: <code>AP_DATA_TOK_REQ</code> , <code>AP_DATA_TOK_ACPT</code>

The argument `cdata→res` is used to convey the result of the negotiation and must be set to one of the following:

Value	Description
<code>AP_ACCEPT</code>	Accept the association.
<code>AP_REJ_PERM</code>	Association permanently rejected.
<code>AP_REJ_TRAN</code>	Association temporarily rejected.

The argument `cdata→diag` is used to indicate the reason for the result specified by `cdata→res` if the negotiation was rejected. If the result was `AP_ACCEPT`, this argument is ignored.

If the result was not AP_ACCEPT, the possible values for *cdata*→*diag* are as follows:

Value	Description
AP_CLD_AEID_NREC	Called AE invocation identifier not recognized.
AP_CLD_AEQ_NREC	Called AE qualifier not recognized.
AP_CLD_APID_NREC	Called AP invocation identifier not recognized.
AP_CLD_APT_NREC	Called AP title not recognized.
AP_CLD_PADDR_UNK	Called presentation address unknown.
AP_CLG_AEID_NREC	Calling AE invocation identifier not recognized.
AP_CLG_AEQ_NREC	Calling AE qualifier not recognized.
AP_CLG_APID_NREC	Calling AP invocation identifier not recognized.
AP_CLG_APT_NREC	Calling AP title not recognized.
AP_CNTX_NAME_NSUP	Calling application context name not supported.
AP_DFCN_NSUP	Default context not supported.
AP_LCL_LIM_EXCEEDED	Local limit exceeded.
AP_NRSN	No reason given.
AP_NULL	Null.
AP_TEMP_CONGESTION	Temporary congestion.
AP_UDATA_NREAD	User data not readable.

If *cdata*→*diag* is set to AP_CLD_PADDR_UNK, AP_LCL_LIM_EXCEEDED, AP_DFCN_NSUP, AP_TEMP_CONGESTION, or AP_UDATA_NREAD the confirmation received by the remote service user will indicate that the source of the association rejection was the presentation service provider (see “A_ASSOC_CNF—confirm association request” on page 106). In this case, any association-information passed with the A_ASSOC_RSP primitive is ignored.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the *ap_error()* function (see page 47).

Errors

AP_AGAIN, AP_LOOK, AP_BADENV, AP_BADLSTATE, BAD_PRIM,
BAD_UBUF, BAD_HANGUP, AP_NOENV, AP_BADCD_DIAG,
AP_BADCD_RES, AP_BADCD_TOKENS, AP_BADSFU

A_ASSOC_CNF—*confirm association request*

Name

A_ASSOC_CNF—used to confirm an association request.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns A_ASSOC_CNF */
    ap_cdata_t   *cdata,  /* ptr to type a_assoc_cnf_cd_t */
    struct osi_buf *ubuf,
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The A_ASSOC_CNF primitive is used in conjunction with ap_rcv() and the APLI environment to confirm a request to establish an association between two application entities.

To receive an association confirmation, the pointer *sptype* must point to an **unsigned long** and the pointer *cdata* must point to a structure of type **a_assoc_cnf_cd_t**, which is defined as follows:

```
typedef struct {
    long    res;           /* result of negotiation */
    long    res_src;       /* source of result */
    long    diag;          /* reason (if rejected) */
    ulong   tokens;        /* initial token assignment */
    long    udata_length;  /* length of user data */
} a_assoc_cnf_cd_t;
```

On return, the **unsigned long** pointed to by *sptype* is set to `A_ASSOC_CNF` and any additional protocol information is returned in the structure pointed to by *cdata*.

The value of *cdata*→*udata_length* is set to indicate the total number of octets of user data received with the primitive in the **osi_buf** structure pointed to by *ubuf*.

The value of *cdata*→*tokens* is set to convey the initial token setting proposed by the initiator, which is defined by OR'ing the values for each available token selected from the following table:

Token	Allowed Values
data	one of: <code>AP_DATA_TOK_REQ</code> , <code>AP_DATA_TOK_ACPT</code>
synchronize_minor	one of: <code>AP_DATA_TOK_REQ</code> , <code>AP_DATA_TOK_ACPT</code>
major/activity	one of: <code>AP_DATA_TOK_REQ</code> , <code>AP_DATA_TOK_ACPT</code>
release	one of: <code>AP_DATA_TOK_REQ</code> , <code>AP_DATA_TOK_ACPT</code>

The value *cdata*→*res* is set to indicate the result of the negotiation as follows:

Value	Description
<code>AP_ACCEPT</code>	Accept the association.
<code>AP_REJ_PERM</code>	Association permanently rejected.
<code>AP_REJ_TRAN</code>	Association temporarily rejected.

The value *cdata*→*res_src* indicates the source of the result as follows:

Value	Description
<code>AP_ACSE_SERV_USER</code>	ACSE service user.
<code>AP_ACSE_SERV_PROV</code>	ACSE service provider.
<code>AP_PRES_SERV_PROV</code>	Presentation service provider.

The argument *cdata*→*diag* is ignored when the value of *cdata*→*res* is `AP_ACCEPT`. When the value of *cdata*→*res* is either `AP_REJ_PERM` or `AP_REJ_TRAN`, the possible values for this argument depend upon the source of the result.

If the source of the result (*cdata*→*res_src*) is AP_ACSE_SERV_USER, *cdata*→*diag* is set to one of the following values:

Value	Description
AP_CLD_AEID_NREC	Called AE invocation identifier not recognized.
AP_CLD_AEQ_NREC	Called QE qualifier not recognized.
AP_CLD_APID_NREC	Called AP invocation identifier not recognized.
AP_CLD_APT_NREC	Called AP title not recognized.
AP_CLG_AEID_NREC	Calling AE invocation identifier not recognized.
AP_CLG_AEQ_NREC	Calling AE invocation identifier not recognized.
AP_CLG_APID_NREC	Calling AP invocation identifier not recognized.
AP_CLG_APT_NREC	Calling AP title not recognized.
AP_NRSN	No reason given.
AP_NULL	Null.

If the source of the result (*cdata*→*res_src*) is AP_ACSE_SERV_PROV, *cdata*→*diag* is set to one of the following values:

Value	Description
AP_NCMN_ACSE_VER	No common version of ACSE protocol supported.
AP_NRSN	No reason given.
AP_NULL	Null.

If the source of the result (*cdata*→*res_src*) is AP_PRES_SERV_PROV, *cdata*→*diag* is set to one of the following values:

Value	Description
AP_CLD_PADDR_UNK	Called presentation address unknown.
AP_DFCN_NSUP	Default context not supported.
AP_DIAG_NOVAL	No value was specified for this parameter.
AP_LCL_LIM_EXCEEDED	Local limit exceeded.
AP_NO_PSAP_AVAIL	No PSAP available.

Value	Description
AP_NRSN	Reason not specified.
AP_NULL	Null.
AP_PRES_VER_NSUP	Protocol version not supported.
AP_SESS_PROV	Could not establish session connection.
AP_TEMP_CONGESTION	Temporary congestion.
AP_UDATA_NREAD	User data not readable.

The value of *cdata*→*tokens* conveys the assignment of tokens that were identified as “responder’s choice” by the original A_ASSOC_REQ and set by the corresponding A_ASSOC_RSP. See “A_ASSOC_RSP—respond to association request” on page 102.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

AP_AGAIN, AP_LOOK, AP_PDUREJ, AP_BADLSTATE, AP_NOBUF

A_PABORT_REQ—*request abnormal provider release*

Name

A_PABORT_REQ—used to initiate a presentation provider abort.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to A_PABORT_REQ */
    ap_cdata_t   *cdata, /* ptr to type a_pabort_req_cd_t */
    struct osi_buf *ubuf,
    int          flags
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function ap_snd().

Description

The A_PABORT_REQ primitive is used in conjunction with ap_snd() and the APLI environment to initiate a presentation layer provider abort. This facility gives the service user the option of aborting an association from the presentation provider when an invalid PDU encoding is encountered.

To issue a provider abort request, the symbolic constant *sptype* must be set to A_PABORT_REQ and the pointer *cdata* must point to a structure of type **a_pabort_req_cd_t**, which is defined as follows:

```
typedef struct {
    long   rsn; /* reason for abort*/
    long   evt; /* event that caused abort */
} a_pabort_req_cd_t;
```


Note that since no user-information is sent with a `A_PABORT_REQ`, the pointer `ubuf` should always be null. The argument `flags` is not used.

The argument `cdata→rsn` is set to indicate the reason for the abort request. The possible values for `cdata→rsn` are as follows:

Value	Description
<code>AP_INVALID_PPDU_PARM</code>	Invalid presentation protocol data unit parameter.
<code>AP_NSPEC</code>	The reason for the abort is not specified.
<code>AP_RSN_NOVAL</code>	No value was supplied for this optional parameter.
<code>AP_UNEXPT_PPDU</code>	Unexpected presentation protocol data unit.
<code>AP_UNEXPT_PPDU_PARM</code>	Unexpected presentation protocol data unit parameter.
<code>AP_UNEXPT_SSPRIM</code>	Unexpected session service primitive.
<code>AP_UNREC_PPDU</code>	Unrecognized presentation protocol data unit.
<code>AP_UNREC_PPDU_PARM</code>	Unrecognized presentation protocol data unit parameter.

The `cdata→evt` field must be set to indicate which PPDU or session primitive triggered the abort. The possible values for `cdata→evt` are as follows:

Value	Description
<code>AP_EVT_NOVAL</code>	No value supplied for this optional parameter.
<code>AP_PEI_AC</code>	Alter context PPDU.
<code>AP_PEI_ACA</code>	Alter context acknowledge PPDU.
<code>AP_PEI_ARP</code>	Abnormal release provider PPDU.
<code>AP_PEI_ARU</code>	Abnormal release user PPDU.
<code>AP_PEI_CP</code>	Connect presentation PPDU.
<code>AP_PEI_CPA</code>	Connect presentation accept PPDU.
<code>AP_PEI_CPR</code>	Connect presentation reject PPDU.
<code>AP_PEI_RS</code>	Resynchronize PPDU.
<code>AP_PEI_RSA</code>	Resynchronize acknowledge PPDU.
<code>AP_PEI_S_ACT_START_IND</code>	Session activity start indication.
<code>AP_PEI_S_ACT_RESUME_IND</code>	Session activity resume indication.

Value	Description
AP_PEI_S_ACT_INT_IND	Session activity interrupt indication.
AP_PEI_S_ACT_INT_CNF	Session activity interrupt confirmation.
AP_PEI_S_ACT_DISC_IND	Session activity discard indication.
AP_PEI_S_ACT_DISC_CNF	Session activity discard confirmation.
AP_PEI_S_ACT_END_IND	Session activity end indication.
AP_PEI_S_ACT_END_CNF	Session activity end confirmation.
AP_PEI_S_CTRLGIVE_IND	Session control give indication.
AP_PEI_S_P_EXCEPT_REP_IND	Session provider exception report indication.
AP_PEI_S_U_EXCEPT_REP_IND	Session user exception report indication.
AP_PEI_S_RELEASE_IND	Session release indication.
AP_PEI_S_RELEASE_CNF	Session release confirmation.
AP_PEI_S_SYNCMAJOR_IND	Session synchronize-major indication.
AP_PEI_S_SYNCMAJOR_CNF	Session synchronize-major confirmation.
AP_PEI_S_SYNCMINOR_IND	Session synchronize-minor indication.
AP_PEI_S_SYNCMINOR_CNF	Session synchronize-minor confirmation.
AP_PEI_S_TOKENGIVE_IND	Session token give indication.
AP_PEI_S_TOKENPLEASE_IND	Session token please indication.
AP_PEI_TC	Capability data PPDU.
AP_PEI_TCC	Capability data acknowledge PPDU.
AP_PEI_TD	Presentation data PPDU.
AP_PEI_TE	Expedited data PPDU.
AP_PEI_TTD	Presentation typed data PPDU.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

AP_AGAIN, AP_LOOK, AP_BADENV, AP_BADLSTATE, BAD_PRIM,
BAD_UBUF, BAD_HANGUP, AP_NOENV, AP_BADCD_EVT, AP_BADCD_RSN

A_PABORT_IND—*indicate abnormal provider release*

Name

A_PABORT_IND—used to indicate a presentation provider abort.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns A_PABORT_IND */
    ap_cdata_t   *cdata,  /* ptr to type a_pabort_ind_cd_t */
    struct osi_buf *ubuf,
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The A_PABORT_IND primitive is used in conjunction with ap_rcv() and the APLI environment to indicate that an association has been abnormally released due to problems in services below the application layer.

To receive a provider abort, the pointer *sptype* must point to an **unsigned long** and the pointer *cdata* must point to a structure of type **a_pabort_ind_cd_t**, which is defined as follows:

```
typedef struct {
    long   rsn; /* reason for abort */
    long   evt; /* event that caused abort */
} a_pabort_ind_cd_t;
```

On return, the **unsigned long** pointed to by *sptype* is set to `A_PABORT_IND` and any additional protocol information is returned in the structure pointed to by *cdata*.

Note that since no user-information is sent with a `A_PABORT_REQ`, the structure **osi_buf** pointed to by the pointer *ubuf* is not updated. The argument *flags* is not used.

The value of *cdata*→*rsn* is set to indicate the reason for the abort request.

Value	Description
<code>AP_INVALID_PPDU_PARM</code>	Invalid presentation protocol data unit parameter.
<code>AP_NSPEC</code>	The reason for the abort is not specified.
<code>AP_RSN_NOVAL</code>	No value was supplied for this optional parameter.
<code>AP_UNEXPT_PPDU</code>	Unexpected presentation protocol data unit.
<code>AP_UNEXPT_PPDU_PARM</code>	Unexpected presentation protocol data unit parameter.
<code>AP_UNEXPT_SSPRIM</code>	Unexpected session service primitive.
<code>AP_UNREC_PPDU</code>	Unrecognized presentation protocol data unit.
<code>AP_UNREC_PPDU_PARM</code>	Unrecognized presentation protocol data unit parameter.

The value of *cdata*→*evt* is set to indicate the event that caused the abort.

Value	Description
<code>AP_EVT_NOVAL</code>	No value supplied for this optional parameter.
<code>AP_PEI_AC</code>	Alter context PPDU.
<code>AP_PEI_ACA</code>	Alter context acknowledge PPDU.
<code>AP_PEI_ARP</code>	Abnormal release provider PPDU.
<code>AP_PEI_ARU</code>	Abnormal release user PPDU.
<code>AP_PEI_CP</code>	Connect presentation PPDU.
<code>AP_PEI_CPA</code>	Connect presentation accept PPDU.
<code>AP_PEI_CPR</code>	Connect presentation reject PPDU.
<code>AP_PEI_RS</code>	Resynchronize PPDU.
<code>AP_PEI_RSA</code>	Resynchronize acknowledge PPDU.

Value	Description
AP_PEI_S_ACT_START_IND	Session activity start indication.
AP_PEI_S_ACT_RESUME_IND	Session activity resume indication.
AP_PEI_S_ACT_INT_IND	Session activity interrupt indication.
AP_PEI_S_ACT_INT_CNF	Session activity interrupt confirmation.
AP_PEI_S_ACT_DISC_IND	Session activity discard indication.
AP_PEI_S_ACT_DISC_CNF	Session activity discard confirmation.
AP_PEI_S_ACT_END_IND	Session activity end indication.
AP_PEI_S_ACT_END_CNF	Session activity end confirmation.
AP_PEI_S_CTRLGIVE_IND	Session control give indication.
AP_PEI_S_P_EXCEPT_REP_IND	Session provider exception report indication.
AP_PEI_S_U_EXCEPT_REP_IND	Session user exception report indication.
AP_PEI_S_RELEASE_IND	Session release indication.
AP_PEI_S_RELEASE_CNF	Session release confirmation.
AP_PEI_S_SYNCMAJOR_IND	Session synchronize-major indication.
AP_PEI_S_SYNCMAJOR_CNF	Session synchronize-major confirmation.
AP_PEI_S_SYNCMINOR_IND	Session synchronize-minor indication.
AP_PEI_S_SYNCMINOR_CNF	Session synchronize-minor confirmation.
AP_PEI_S_TOKENGIVE_IND	Session token give indication.
AP_PEI_S_TOKENPLEASE_IND	Session token please indication.
AP_PEI_TC	Capability data PPDU.
AP_PEI_TCC	Capability data acknowledge PPDU.
AP_PEI_TD	Presentation data PPDU.
AP_PEI_TE	Expedited data PPDU.
AP_PEI_TTD	Presentation typed data PPDU.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

AP_AGAIN, AP_LOOK, AP_PDUREJ, AP_BADLSTATE, AP_NOBUF

A_RELEASE_REQ—*request normal release*

Name

A_RELEASE_REQ—used to request the release of an association.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to A_RELEASE_REQ */
    ap_cdata_t   *cdata, /* ptr to type a_release_req_cd_t */
    struct osi_buf *ubuf,
    int          flags
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function ap_snd().

Description

The A_RELEASE_REQ primitive is used in conjunction with ap_snd() and the APLI environment to request the normal release of an association.

To issue a release request, the symbolic constant *sptype* must be set to A_RELEASE_REQ and the pointer *cdata* must point to a structure of type **a_release_req_cd_t**, which is defined as follows:

```
typedef struct {
    long   rsn; /* reason for release */
    long   udata_length; /* length of user-info */
} a_release_req_cd_t;
```


If the primitive is issued using multiple calls to `ap_snd()` with the `AP_MORE` flag set, the argument `cdata→udata_length` must be set to the total number of octets of user information to be sent in the `osi_buf` structure pointed to by `ubuf`. This argument is ignored if the primitive is issued as a single `ap_snd()`.

The argument `cdata→rsn` is used to convey the reason for the release request. Providing a reason for the release request is optional. Valid values are as follows:

Value	Description
<code>AP_REL_NORMAL</code>	Normal release request.
<code>AP_REL_URGENT</code>	Urgent release request.
<code>AP_REL_USER_DEF</code>	A user defined release request.
<code>AP_RSN_NOVAL</code>	No value was specified for this optional parameter.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `ap_errno` is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_BADENV`, `AP_BADLSTATE`, `BAD_PRIM`,
`BAD_UBUF`, `BAD_HANGUP`, `AP_NOENV`, `AP_BADCD_RSN`

A_RELEASE_IND—*indicate normal release request*

Name

A_RELEASE_IND—used to indicate an association release request.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns A_RELEASE_IND */
    ap_cdata_t   *cdata,  /* ptr to type a_release_ind_cd_t */
    struct osi_buf *ubuf,
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The A_RELEASE_IND primitive is used in conjunction with ap_rcv() and the APLI environment to indicate that the remote service user wishes to release the association.

To receive a release indication, the pointer *sptype* must point to an **unsigned long** and the pointer *cdata* must point to a structure of type **a_release_ind_cd_t**, which is defined as follows:

```
typedef struct {
    long    rsn;           /* reason for release */
    long    udata_length;  /* length of user-info */
} a_release_ind_cd_t;
```

On return, the **unsigned long** pointed to by *sptype* is set to `A_RELEASE_IND` and any additional protocol information is returned in the structure pointed to by *cdata*.

The value of *cdata*→*udata_length* is set to indicate the total number of octets of user data received with the primitive in the **osi_buf** structure pointed to by *ubuf*.

The value of *cdata*→*rsn* is set to indicate the reason for the release request, provided this optional parameter was set by the `A_RELEASE_REQ` primitive. The possible values for *cdata*→*rsn* are as follows:

Value	Description
<code>AP_REL_NORMAL</code>	Normal release request.
<code>AP_REL_URGENT</code>	Urgent release request.
<code>AP_REL_USER_DEF</code>	A user defined release request.
<code>AP_RSN_NOVAL</code>	No value was specified for this optional parameter.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_PDUREJ`, `AP_BADLSTATE`, `AP_NOBUF`

A_RELEASE_RSP—*respond to normal release request*

Name

A_RELEASE_RSP—used to respond to an association release request.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to A_RELEASE_RSP */
    ap_cdata_t   *cdata, /* ptr to type a_release_rsp_cd_t */
    struct osi_buf *ubuf,
    int          flags
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function ap_snd().

Description

The A_RELEASE_RSP primitive is used in conjunction with ap_snd() and the APLI environment to respond to a normal association release request.

To issue a release response, the symbolic constant *sptype* must be set to A_RELEASE_RSP and the pointer *cdata* must point to a structure of type **a_release_rsp_cd_t**, which is defined as follows:

```
typedef struct {
    long  res;           /* result of negotiation */
    long  rsn;           /* reason for release */
    long  udata_length;  /* length of user-info */
} a_release_rsp_cd_t;
```

If the primitive is issued using multiple calls to `ap_snd()` with the `AP_MORE` flag set, the argument `cdata→udata_length` must be set to the total number of octets of user information to be sent in the `osi_buf` structure pointed to by `ubuf`. This argument is ignored if the primitive is issued as a single `ap_snd()`.

The argument `cdata→res` is used to convey the result of the negotiation. It must be set to one of the following values.

Value	Description
<code>AP_REL_AFFIRM</code>	Indicates acceptance of the release.
<code>AP_REL_NEGATIVE</code>	Indicates rejection of the release. Note that this value can only be used if the session negotiated release functional unit has been negotiated.

The argument `cdata→rsn` is used to convey the reason for the result returned in `cdata→res`. It is an optional parameter that can be set to one of the following values:

Value	Description
<code>AP_REL_NORMAL</code>	Indicates a normal release.
<code>AP_REL_NOTFINISHED</code>	Indicates the user is not finished with the association.
<code>AP_REL_USER_DEF</code>	Indicates a user defined reason.
<code>AP_RSN_NOVAL</code>	No value was supplied for this optional parameter.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `ap_errno` is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_BADENV`, `AP_BADLSTATE`, `BAD_PRIM`,
`BAD_UBUF`, `BAD_HANGUP`, `AP_NOENV`

A_RELEASE_CNF—*confirm normal release request*

Name

A_RELEASE_CNF—used to confirm an association release request.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns A_RELEASE_CNF */
    ap_cdata_t   *cdata,  /* ptr to type a_release_cnf_cd_t */
    struct osi_buf *ubuf,
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The A_RELEASE_CNF primitive is used in conjunction with ap_rcv() and the APLI environment to confirm the acceptance or rejection of the association release request.

To receive a release confirmation, the pointer *sptype* must point to an **unsigned long** and the pointer *cdata* must point to a structure of type **a_release_cnf_cd_t**, which is defined as follows:

```
typedef struct {
    long   res;           /* result of release request */
    long   rsn;           /* reason for release */
    long   udata_length;  /* length of user-info */
} a_release_cnf_cd_t;
```

On return, the **unsigned long** pointed to by *sptype* is set to `A_RELEASE_CNF` and any additional protocol information is returned in the structure pointed to by *cdata*.

The value of *cdata*→*udata_length* is set to indicate the total number of octets of user data received with the primitive in the **osi_buf** structure pointed to by *ubuf*.

The value of *cdata*→*res* is set to indicate the result of the release negotiation. It is set to one of the following values:

Value	Description
<code>AP_REL_AFFIRM</code>	Indicates acceptance of the release.
<code>AP_REL_NEGATIVE</code>	Indicates rejection of the release. Note that this value can only be used if the session negotiated release functional unit has been negotiated.

The value of *cdata*→*rsn* is set to indicate the reason for the result returned by *cdata*→*res*, provided this optional parameter was set by the `A_RELEASE_RSP` primitive. The possible values for *cdata*→*rsn* are as follows:

Value	Description
<code>AP_REL_NORMAL</code>	Indicates a normal release.
<code>AP_REL_NOTFINISHED</code>	Indicates the user is not finished with the association.
<code>AP_REL_USER_DEF</code>	Indicates a user defined reason.
<code>AP_RSN_NOVAL</code>	No value was supplied for this optional parameter.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_PDUREJ`, `AP_BADLSTATE`, `AP_NOBUF`

P_DATA_REQ—*send user data*

Name

P_DATA_REQ—used to send normal user data.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to P_DATA_REQ */
    ap_cdata_t   *cdata, /* ptr to type p_data_req_cd_t */
    struct osi_buf *ubuf,
    int          flags,
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function ap_snd().

Description

The P_DATA_REQ primitive is used in conjunction with ap_snd() and the APLI environment to send user data over an established association. It must be issued in association with one or more octets of user data.

Note that the P_DATA_REQ primitive cannot be issued without one or more octets of *user data* passed in the buffer **osi_buf** pointed to by *ubuf*.

To send user data, the symbolic constant *sptype* must be set to P_DATA_REQ and the pointer *cdata* must point to a structure of type **p_data_req_cd_t**, which is defined as follows:

```
typedef struct {
    ulong tokens; /* token assignment */
} p_data_req_cd_t;
```


The argument *cdata→tokens* is used to identify the tokens to be surrendered. Tokens are identified by OR'ing one or more of the following values:

Value	Description
AP_DATA_TOK	Data token
AP_MAJACT_TOK	Synchronize major/activity token
AP_SYNCMINOR_TOK	Synchronize minor token
AP_RELEASE_TOK	Release token

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

AP_AGAIN, AP_LOOK, AP_BADENV, AP_BADLSTATE, BAD_PRIM,
BAD_UBUF, BAD_HANGUP, AP_NOENV, AP_BADCD_TOKENS, AP_BADUBUF

P_DATA_IND—*indicate receipt of user data*

Name

P_DATA_IND—used to indicate receipt of normal user data.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns P_DATA_IND */
    ap_cdata_t   *cdata,  /* not used */
    struct osi_buf *ubuf,  /* user data */
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The P_DATA_IND primitive is used in conjunction with ap_rcv() and the APLI environment to indicate the receipt of user data.

To receive user data, the pointer *sptype* must point to an **unsigned long**.

On return, the **unsigned long** pointed to by *sptype* is set to P_DATA_IND. The user data sent by the initiator is received in the buffer **osi_buf** pointed to by *ubuf*. The structure pointed to by *cdata* is not used.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the ap_error() function (see page 47).

Errors

AP_AGAIN, AP_LOOK, AP_PDUREJ, AP_BADLSTATE, AP_NOBUF

P_TOKENGIVE_REQ—*give session token*

Name

P_TOKENGIVE_REQ—used to give a token.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to P_TOKENGIVE_REQ */
    ap_cdata_t    *cdata, /* ptr to type p_tokengive_req_cd_t */
    struct osi_buf *ubuf,
    int          flags
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function ap_snd().

Description

The P_TOKENGIVE_REQ primitive is used in conjunction with ap_snd() and the APLI environment to provide the user with access to the S-TOKEN-GIVE service of the session layer.

The S-TOKEN-GIVE session service allows one session service user to surrender one or more tokens to the other session service user. A token is an attribute of an association which is dynamically assigned to one user at a time. The user that currently possesses a token has the exclusive use of the service it controls.

Note that when the P_TOKENGIVE_REQ primitive is issued using multiple ap_snd() calls (with the AP_MORE bit set), there must be one or more octets of user-information associated with the final ap_snd() call of the sequence.

User-information may not be sent in association with the P_TOKENGIVE_REQ primitive when **Session Version 1** is in effect.

To surrender tokens, the symbolic constant *sptype* must be set to `P_TOKENGIVE_REQ` and the pointer *cdata* must point to a structure of type `p_tokengive_req_cd_t`, which is defined as follows:

```
typedef struct {
    ulong  tokens;           /* token assignment */
    long   udata_length;     /* length of user data */
} p_tokengive_req_cd_t;
```

If the primitive is issued using multiple calls to `ap_snd()` with the `AP_MORE` flag set, the argument `cdata→udata_length` must be set to the total number of octets of user information to be sent in the `osi_buf` structure pointed to by *ubuf*. This argument is ignored if the primitive is issued as a single `ap_snd()`.

The argument `cdata→tokens` is used to convey the tokens to be surrendered. Tokens are identified by OR'ing one or more of the following values:

Value	Description
<code>AP_DATA_TOK</code>	Data token
<code>AP_MAJACT_TOK</code>	Synchronize major/activity token
<code>AP_SYNCMINOR_TOK</code>	Synchronize minor token
<code>AP_RELEASE_TOK</code>	Release token

In addition, the APLI environment attribute `AP_TOKENS_OWNED` will be updated to reflect the exchange of tokens.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_BADENV`, `AP_BADLSTATE`, `BAD_PRIM`,
`BAD_UBUF`, `BAD_HANGUP`, `AP_NOENV`

P_TOKENGIVE_IND—*indicate receipt of token*

Name

P_TOKENGIVE_IND—used to indicate receipt of newly acquired tokens.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns P_TOKENGIVE_IND */
    ap_cdata_t   *cdata, /* ptr to type p_tokengive_ind_cd_t */
    struct osi_buf *ubuf,
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The P_TOKENGIVE_IND primitive is used in conjunction with ap_rcv() and the APLI environment to indicate the receipt of newly acquired tokens. A token is an attribute of an association which is dynamically assigned to one user at a time. The user that currently possesses a token has the exclusive use of the service it controls.

To accept surrendered tokens, the pointer *sptype* must point to an **unsigned long** and the pointer *cdata* must point to a structure of type **p_tokengive_ind_cd_t**, which is defined as follows:

```
typedef struct {
    ulong tokens;           /* token assignment */
    long  udata_length;     /* length of user data */
} p_tokengive_ind_cd_t;
```

On return, the **unsigned long** pointed to by *sptype* is set to `P_TOKENGIVE_IND` and any additional protocol information is returned in the structure pointed to by *cdata*.

The value of *cdata*→*udata_length* is set to indicate the total number of octets of user data received with the primitive in the **osi_buf** structure pointed to by *ubuf*.

The value of *cdata*→*tokens* is set to identify the tokens received from the initiator. Tokens are identified by OR'ing one or more of the following values:

Value	Description
<code>AP_DATA_TOK</code>	Data token
<code>AP_MAJACT_TOK</code>	Synchronize major/activity token
<code>AP_SYNCMINOR_TOK</code>	Synchronize minor token
<code>AP_RELEASE_TOK</code>	Release token

In addition, the APLI environment attribute `AP_TOKENS_OWNED` will be updated to reflect the exchange of tokens.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_PDUREJ`, `AP_BADLSTATE`, `AP_NOBUF`

P_TOKENPLEASE_REQ—*request token*

Name

P_TOKENPLEASE_REQ—used to request a session token.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_snd (
    int          fd,
    unsigned long sptype, /* set to P_TOKENPLEASE_REQ */
    ap_cdata_t    *cdata, /* ptr to type p_tokenplease_req_cd_t */
    struct osi_buf *ubuf,
    int          flags
);
```

Arguments

See “ap_snd()—send primitive” on page 37 for a detailed description of the arguments passed to the function ap_snd().

Description

The P_TOKENPLEASE_REQ primitive is used in conjunction with ap_snd() and the APLI environment to provide the user with access to the S-TOKEN-PLEASE service of the session layer.

The S-TOKEN-PLEASE session service allows one session service user to request one or more tokens from the other session service user. A token is an attribute of an association which is dynamically assigned to one user at a time. The user that currently possesses a token has the exclusive use of the service it controls.

Note that when the P_TOKENPLEASE_REQ primitive is issued using multiple ap_snd() calls (with the AP_MORE bit set), there must be one or more octets of user-information associated with the final ap_snd() call of the sequence.

To request tokens, the symbolic constant *sptype* must be set to `P_TOKENPLEASE_REQ` and the pointer *cdata* must point to a structure of type `p_tokenplease_req_cd_t`, which is defined as follows:

```
typedef struct {
    ulong  tokens;           /* token assignment */
    long   udata_length;     /* length of user data */
} p_tokenplease_req_cd_t;
```

If the primitive is issued using multiple calls to `ap_snd()` with the `AP_MORE` flag set, the argument `cdata→udata_length` must be set to the total number of octets of user information to be sent in the `osi_buf` structure pointed to by *ubuf*. This argument is ignored if the primitive is issued as a single `ap_snd()`.

The argument `cdata→tokens` is used to identify the tokens requested. Tokens are identified by OR'ing one or more of the following values:

Value	Description
<code>AP_DATA_TOK</code>	Data token
<code>AP_MAJACT_TOK</code>	Synchronize major/activity token
<code>AP_SYNCMINOR_TOK</code>	Synchronize minor token
<code>AP_RELEASE_TOK</code>	Release token

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_BADENV`, `AP_BADLSTATE`, `BAD_PRIM`,
`BAD_UBUF`, `BAD_HANGUP`, `AP_NOENV`, `AP_BADCD_TOKENS`, `AP_NODATA`

P_TOKENPLEASE_IND—*indicate request for token*

Name

P_TOKENPLEASE_IND—used to indicate request for tokens.

Synopsis

```
#include <ap_lib.h>
#include <osi_lib.h>

int ap_rcv (
    int          fd,
    unsigned long *sptype, /* returns P_TOKENPLEASE_IND */
    ap_cdata_t    *cdata, /* ptr to type p_tokenplease_ind_cd_t */
    struct osi_buf *ubuf,
    int          *flags
);
```

Arguments

See “ap_rcv()—receive primitive” on page 40 for a detailed description of the arguments passed to the function ap_rcv().

Description

The P_TOKENPLEASE_IND primitive is used in conjunction with ap_rcv() and the APLI environment to indicate the receipt of a request for tokens. A token is an attribute of an association which is dynamically assigned to one user at a time. The user that currently possesses a token has the exclusive use of the service it controls.

To accept a request for surrendered tokens, the pointer *sptype* must point to an **unsigned long** and the pointer *cdata* must point to a structure of type **p_tokenplease_ind_cd_t**, which is defined as follows:

```
typedef struct {
    ulong tokens;           /* token assignment */
    long  udata_length;     /* length of user data */
} p_tokenplease_ind_cd_t;
```

On return, the **unsigned long** pointed to by *sptype* is set to `P_TOKENPLEASE_IND` and any additional protocol information is returned in the structure pointed to by *cdata*.

The value of *cdata*→*udata_length* is set to indicate the total number of octets of user data received with the primitive in the **osi_buf** structure pointed to by *ubuf*.

The value of *cdata*→*tokens* is set to identify the tokens surrendered to the initiator. Tokens are identified by OR'ing one or more of the following values:

Value	Description
<code>AP_DATA_TOK</code>	Data token
<code>AP_MAJACT_TOK</code>	Synchronize major/activity token
<code>AP_SYNCMINOR_TOK</code>	Synchronize minor token
<code>AP_RELEASE_TOK</code>	Release token

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *ap_errno* is set to indicate the error. The corresponding error message is returned by calling the `ap_error()` function (see page 47).

Errors

`AP_AGAIN`, `AP_LOOK`, `AP_PDUREJ`, `AP_BADLSTATE`, `AP_NOBUF`

APLI Example Application



<i>Example Files</i>	<i>page 140</i>
<i>Modifying the Local Environment File (enviro.h)</i>	<i>page 141</i>
<i>Compiling and Linking the Example Code</i>	<i>page 143</i>
<i>Running the Example Application</i>	<i>page 143</i>

This appendix describes the files associated with the example APLI application provided on the *SunLink OSI 8.1 Communication Platform* CD-ROM and describes how to modify the application to run in your local environment.

The example application presented is intended solely as a general description of how the SunLink implementation of APLI might be used. It should not be construed as an attempt to provide a template for constructing any particular application. Moreover, it is assumed that the service user is familiar with the ACSE and presentation layer protocols and understands the role of the ACSE service user in establishing, using, and terminating an association between two application entities.

Example Files

If you installed SunLink OSI using the default base directories, the following files are located in the directory `/opt/SUNWconn/osinet/example`:

<code>Makefile</code>	Makefile used to compile and link example code.
<code>enviro.h</code>	Header file that defines the local environment.
<code>utils.c</code>	Application utilities called by example application.
<code>ex_initiator.c</code>	Initiator portion of example application.
<code>ex_responder.c</code>	Responder portion of example application.

Modifying the Local Environment File (enviro.h)

To use the example application you must modify the contents of the header file `enviro.h` to enter the addresses that correspond to your local environment. Addresses can be entered in hexadecimal or character (ASCII) format.

```

/*****/
/** Environment value to run APLI under RFC1006      */
/** In this case you can try in LOOPBACK mode      */
/*****/
#ifdef RFC1006
/** Declaration for the initiator side */
char *IniPresSel="pre";           /* defined using ositool */
char *IniSesSel="prs";           /* defined using ositool */
char *IniTrpSel=0x0;             /* Transport Selector must be zero */
char *IniNsap="0x819db37b";      /* IP Address */
char *IniBindNsap="";           /* Bind Address must be zero */

/** Declaration for the responder side (loopback) */
char *ResPresSel="prs";          /* defined using ositool */
char *ResSesSel="pre";           /* defined using ositool */
char *ResTrpSel=0x0;             /* Transport Selector must be zero */
char *ResNsap="0x819db37b";      /* IP Address */
char *ResBindNsap="";           /* Bind Address must be zero */
#endif

/*****/
/** Environment value to run APLI under LAN          */
/** In this case you can try in LOOPBACK mode      */
/*****/
#ifdef LAN
/** Declaration for the initiator side */
char *IniPresSel="pre";          /* defined using ositool */
char *IniSesSel="prs";           /* defined using ositool */
char *IniTrpSel="ses";           /* defined using ositool */
char *IniNsap="lore";            /* NSAP name defined using ositool */
char *IniBindNsap="lore";        /* NSAP name defined using ositool */

/*
char *IniNsap="0x08002011123e";    /* hex equiv of NSAP name */
char *IniBindNsap="0x08002011123e"; /* hex equiv of NSAP name */
*/

```

```

/** Declaration for the responder side (loopback) */
char *ResPresSel="pre";      /* defined using ositool */
char *ResSesSel="prs";      /* defined using ositool */
char *ResTrpSel="ses";      /* defined using ositool */
char *ResNsap="lore";      /* NSAP name defined using ositool */
char *ResBindNsap="lore";   /* NSAP name defined using ositool */

/*
char *ResNsap="0x08002011123e"; /* hex equiv of NSAP name */
char *ResBindNsap="0x08002011123e"; /* hex equiv of NSAP name */
*/
#endif

/*****
*** Environment value to run APLI under X25 ***
*** This compilation doesn't work in LOOPBACK ***
*****/
#ifdef X25
/** Declaration for the initiator side */
char *IniPresSel="pre";      /* defined using ositool */
char *IniSesSel="prs";      /* defined using ositool */
char *IniTrpSel="ses";      /* defined using ositool */
char *IniNsap="0x370000000987654abcdef01"; /* X25 address */
char *IniBindNsap="0x370000000987654abcdef01"; /* X25 address */

/** Declaration for the responder side */
char *ResPresSel="prs";      /* defined using ositool */
char *ResSesSel="pre";      /* defined using ositool */
char *ResTrpSel="ses";      /* defined using ositool */
char *ResNsap="0x3700000012345678abcdef01"; /* X25 address */
char *ResBindNsap="0x3700000012345678abcdef01"; /* X25 address */
#endif

```

Note that you must install and configure your X.25 network in order to run the example application over an X.25 connection.

Compiling and Linking the Example Code

To compile and link the example code using the `Makefile` supplied, you must specify the type of network connection that you want to use. The options are RFC1006, LAN, and X25.

For example, to compile and link the example code so that the application will run over a LAN connection, type:

```
prompt% make LAN
```

Running the Example Application

After you have modified the header file `enviro.h`, and compiled and linked the example code, you can run the resulting application as follows:

1. In one shell tool window (the responder), type:

```
prompt% ex_responder
```

2. In another shell tool window (the initiator), type:

```
prompt% ex_initiator
```

3. Once the connection is established, enter data at the prompt in the initiator shell tool.

Referenced Documents



The following documents are referenced in this book:

The OSI Reference Model

ISO 7498: 1984

Information Processing Systems—Open Systems Interconnection—Basic Reference Model.

ISO/IEC Session, Presentation, ACSE

ISO/IEC pDISP 11188

International Standardized Profiles for ISO/IEC Session, Presentation and ACSE are under development as "ISO/IEC pDISP 11188—Common Upper Layer Requirements—Part 3: Minimal Upper Layer Facilities. This is expected to reach ISP status by Q1/94.

ISO ACSE

ISO 8649: 1988

Information Processing Systems—Open Systems Interconnection—Service Definition for the Association Control Service Element.

ISO 8650: 1988

Information Processing Systems—Open Systems Interconnection—Protocol Specification for the Association Control Service Element

ISO Presentation

ISO 8822: 1988

Information Processing Systems—Open Systems Interconnection—Connection Oriented Presentation Service Definition

ISO 8823: 1988

Information Processing Systems—Open Systems Interconnection—Connection Oriented Presentation Protocol Specification

ASN.1 Notation

ISO/IEC 8824: 1990

Information Technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1)

ASN.1 Basic Encoding Rules

ISO/IEC 8825: 1990

Information Technology—Open Systems Interconnection—Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

OSI Session Layer

ISO 8326: 1987

Information processing systems—Open Systems Interconnection—Connection Oriented Session Service Definition.

ISO 8326/AD 2: 1988

Information processing systems—Open Systems Interconnection—Connection Oriented Session Service Definition—Addendum 2: Incorporation of unlimited user data.

ISO 8326/AMD 4: 1992

Information processing systems—Open Systems Interconnection—Connection Oriented Session Service Definition—Amendment 4: Additional synchronization functionality.

ISO 8327: 1987

Information processing systems—Open Systems Interconnection—Connection Oriented Session Protocol Specification.

ISO 8327/AD 2: 1988

Information processing systems—Open Systems Interconnection—Connection Oriented Session Protocol Specification—Addendum 2: Incorporation of unlimited user data.

ISO 8327/AMD 3: 1992

Information processing systems—Open Systems Interconnection—Connection Oriented Session Protocol Specification—Amendment 3: Additional synchronization functionality.

≡ *B*

Glossary

AARE

A-ASSOCIATE-RESPONSE APDU.

AARQ

A-ASSOCIATE-REQUEST APDU.

abstract syntax

The abstract description of a set of data values. Used by an application-service-element or an application to define the data structures to be transferred. In OSI this is expressed using the ASN.1 abstract syntax notation. The application and presentation layers negotiate the set of abstract syntaxes to be used to transfer data values on an association.

AC

Alter Context PPDU.

AE

Application-entity.

AET

Application-entity-title.

APDU

Application-protocol-data-unit.

ARP

Provider-abort PPDU.

ARU

User-abort PPDU.

application-context

The set of rules that govern the communication between two AEs. These rules include the list of ASEs required to support that communication.

application-entity

(AE) Defined as “The aspects of an application-process pertinent to OSI.” An application may contain one or more AEs, each of which performs part of the OSI-related functions required by the application. For example, a network management application might contain one application entity to access an OSI directory service and another to perform the OSI management functions. An AE communicates with other AEs (using the services of one or more ASEs) to perform its functions.

application-entity-title

(AET) Identifies a particular AE within an application-process. An AET is associated with a single presentation address.

Application layer

Seventh and highest layer in the OSI basic reference model. This layer provides the means by which application processes access the OSI environment. The Application layer is structured as a set of *application-service-elements* that an application may use to access OSI communications capabilities.

application-service-element

(ASE) Defined as “a set of application-functions that provides a capability for the interworking of application-entity-invocations for a specific purpose.” Some ASEs provide generally useful services (e.g., ACSE, the connection management service element), while others provide services oriented to a particular application (e.g., CMISE, the common management information service element).

ASE

Application-service-element.

CP

Presentation-connect PPDU.

CPA

Presentation-connect-accept PPDU.

CPR

Presentation-connect-reject PPDU.

PPDU

Presentation-protocol-data-unit.

presentation context

An association of an **abstract syntax** with a **transfer syntax**, negotiated by the presentation layer when an application association is established. The application layers propose the abstract syntaxes to be used on the association; the presentation layer negotiates the transfer syntaxes to be used to support each of the abstract syntaxes. When transferring data, the application layer identifies the presentation context to be used to encode and decode the data.

Presentation layer

Sixth layer in the OSI reference model. This layer preserves the meaning of the data transferred between AEs. In addition, it provides access to the services of the session layer. Presentation layer functions include syntax negotiation (agreement of the *abstract syntaxes* to be used to transfer data and the *transfer syntaxes* used to encode and decode them), and syntax transformation (from local concrete syntax to transfer syntax and back again).

transfer syntax

The concrete syntax used to transfer data between AEs. For a given **abstract syntax**, the presentation layer negotiates one or more transfer syntaxes that may be used to preserve the meaning of the data during transfer.

Session layer

Fifth layer in the OSI reference model. This layer provides services that allow AEs to organize and synchronize their interactions. In addition to the connection and data transfer services of the transport layer, the session layer provides orderly release, synchronization, activity management, and half-duplex operation.

Transport layer

Fourth layer in the OSI reference model. This layer provides a transparent connection and duplex data transfer service between OSI end systems. transport layer functions include end-to-end sequencing, flow control, error detection, and recovery.

Index

Symbols

/dev/oopi, 9, 15

A

A_ABORT_IND, 10, 95
a_abort_ind_cd_t, 95
A_ABORT_REQ, 10, 93
a_abort_req_cd_t, 93
A_ASSOC_CNF, 10, 63, 106
a_assoc_cnf_cd_t, 106
A_ASSOC_IND, 10, 65, 100
A_ASSOC_REQ, 10, 65, 97
a_assoc_req_cd_t, 97
A_ASSOC_RSP, 10, 63, 102
a_assoc_rsp_cd_t, 102
A_PABORT_IND, 10, 19, 114
a_pabort_ind_cd_t, 114
A_PABORT_REQ, 10, 110
a_pabort_req_cd_t, 110
A_RELEASE_CNF, 10, 124
a_release_cnf_cd_t, 124
A_RELEASE_IND, 10, 120
a_release_ind_cd_t, 120
A_RELEASE_REQ, 10, 118
a_release_req_cd_t, 118

A_RELEASE_RSP, 10, 122
a_release_rsp_cd_t, 122
abnormal
 provider abort, 110, 114
 release, 93, 95
abstract syntax, 4, 19, 149
ACSE, 1, 3
 services, 17
activity service, 5
AE, 149
AE-invocation-identifier, 55, 57, 67
AE-qualifier, 55, 67
AET, 149
any_t, 25, 55, 56, 58, 79
AP_ACSE_AVAIL, 52, 54
AP_ACSE_SEL, 52, 54
AP_ACSEVER1, 54
AP_AGAIN, 20, 41
AP_ASSOC_REQ, 38
AP_BIND_PADDR, 16, 52, 55, 60, 97, 100
ap_cdata_t, 18, 37, 40
ap_cdl_t, 63, 72
ap_cdrl_t, 63, 73
AP_CLD_AEID, 52, 55
AP_CLD_AEID_NOVAL, 55
AP_CLD_AEQ, 52, 55

AP_CLD_APIID, 52, 56	AP_MAX_SYNCNP, 60
AP_CLD_APIID_NOVAL, 56	AP_MIN_SYNCNP, 60
AP_CLD_APT, 52, 56	AP_MODE_AVAIL, 53, 61
AP_CLD_CONN_ID, 52, 56	AP_MODE_SEL, 53, 62
ap_cld_conn_id_t, 56	AP_MORE, 19, 22, 38, 41, 62
AP_CLG_AEID, 52, 57	AP_MSTATE, 53, 62
AP_CLG_AEID_NOVAL, 57	AP_NDELAY, 20, 70
AP_CLG_AEQ, 52, 57	AP_NORMAL_MODE, 61, 62
AP_CLG_APIID, 52, 57	ap_octet_string_t, 78
AP_CLG_APIID_NOVAL, 57	ap_open(), 9, 15
AP_CLG_APT, 52, 58	ap_paddr_t, 55, 60, 65, 78
AP_CLG_CONN_ID, 52, 58	AP_PCDL, 16, 53, 62
ap_clg_conn_id_t, 58	AP_PCDRL, 16, 53, 63
ap_close(), 9, 20, 49	AP_PFU_AVAIL, 53, 64
AP_CNTX_NAME, 52, 58	AP_PFU_SEL, 53, 64
ap_conn_id_t, 75	ap_poll(), 9, 20, 22, 25, 45
AP_DATA_TOK, 71	ap_poll.h, 25
AP_DATA_XFER, 70	ap_pollfd_t, 45
AP_DCN, 52	AP_PRES_AVAIL, 53, 64
ap_dcn_t, 76	AP_PRES_SEL, 53, 65
AP_DCS, 52, 59	AP_PRESVER1, 64
ap_dcs_t, 59, 77	ap_rcv(), 9, 11, 17, 20, 40
AP_DPCN, 16, 59	AP_RCVMORE, 62
AP_DPCR, 16, 53, 60	AP_RELEASE_TOK, 71
AP_DPCR_NOVAL, 60	AP_REM_PADDR, 8, 16, 52, 53, 65, 97
ap_errno, 47	AP_ROLE_ALLOWED, 53, 65
ap_errno(), 22	AP_ROLE_CURRENT, 53, 66
ap_error(), 9, 47	AP_RSP_AEID, 53, 67
ap_free(), 9, 18, 43	AP_RSP_AEQ, 53, 67
ap_get_env(), 9, 15, 51	AP_RSP_APIID, 53, 67
AP_IDLE, 70	AP_RSP_APT, 53, 68
ap_init_env(), 9, 15, 31, 51	AP_SESS_AVAIL, 53, 68
AP_INIT_SYNC_PT, 53, 60	AP_SESS_OPT_AVAIL, 53
AP_INIT_TOKENS, 43	AP_SESS_SEL, 53, 68
AP_INITIATOR, 65, 66	ap_set_env(), 9, 15, 51
AP_LCL_PADDR, 53, 55, 60	AP_SFU_AVAIL, 53, 69
ap_lib.h, 18, 25, 40, 72	AP_SFU_SEL, 53, 69
AP_LIB_AVAIL, 53, 61	ap_snd(), 9, 11, 17, 20
AP_LIB_SEL, 52, 53, 61	AP_SNDMORE, 62
AP_LIBVER1, 61	AP_STATE, 8, 54, 70

- AP_STREAM_FLAGS, 20, 53, 70
- AP_SYNCMAJOR_TOK, 71
- AP_SYNCMINOR_TOK, 71
- AP_TOKENS_AVAIL, 54, 69, 71
- AP_TOKENS_OWNED, 54, 71
- AP_UNBOUND, 70
- AP_WASSOCrsp_ASSOCind, 70
- AP-invocation-identifier, 56, 57, 67
- API
 - environment, 7, 8, 15, 51
 - functions, 7, 9, 27
 - instance, 9, 11, 15
 - instance identifier, 9
 - service primitives, 10, 17, 89
 - service provider, 7, 29
 - service user, 7
 - user data, 19
- application
 - context, 3, 150
 - context name, 58
 - entity, 3, 150
 - entity title, 150
 - layer, 150
 - service element, 2, 150
- AP-qualifier, 57
- AP-title, 56, 58, 68
- ASE, 150
- ASN.1, 4
- association
 - abort indocation, 95
 - abort request, 93
 - control, 3
 - establishment, 21
 - provider abort request, 110
 - release confirmation, 124
 - release indication, 120
 - release request, 118
 - release response, 122
 - request, 97
 - request confirmation, 106
 - request indication, 100
 - request response, 102
- attributes, 15, 51

- default values, 15
 - summary, 52
- auxiliary functions, 13, 22

B

- basic functions, 13
- bind presentation address, 55
- blocking mode, 20, 22, 30

C

- cdata, 37, 40
- closing
 - an API instance, 20, 49
- CMISE, 3
- code
 - example, 139
 - fragments, 27
- communication platform, 1, 9, 11, 25, 29
- compiling API applications, 25, 143
- concurrent API instances, 15, 20, 49
- configuring the API environment, 15
- confirmation, 6
- confirmed negotiation, 6, 17
- connection-oriented network, 2
- context
 - definition list, 16
 - definition response list, 16
 - identifier, 59
- current status, 8
- custom application, 9, 12

D

- data, 21
 - decoding, 19
 - encoding, 19
 - link layer, 2
 - token, 21, 71
 - transfer mechanism, 10, 126, 143
 - transfer service, 5
 - types, 72
- default

- attributes, 15, 31
- context name, 59
- presentation context result, 16, 60
- defined-context-set, 59
- developing applications, 11
- development summary, 12
- device, 9
- dynamically allocated memory, 18, 20

E

- env_file, 31
- enviro.h, 141
- environment, 7, 8, 15, 51
 - file, 141
 - state table, 81
 - summary, 52
- error
 - reporting, 22, 47
 - summary, 23
- establishing an APLI instance, 15, 29
- events, 45
- example application, 13, 26, 27, 139
- exchanging tokens, 130
- execution mode, 20
- external include files, 25

F

- fd, 29, 49
- fds, 45
- file descriptor, 9, 15, 20, 29, 49
- flags, 20, 31
- flowchart, 14
- freeing memory, 18, 20, 22, 43
- FTAM, 3
- functions, 7, 9, 27
 - summary, 28

G

- global variables, 22

H

- half-duplex data transfer, 5
- handshaking, 6, 17
- header file
 - ap_lib.h, 18, 25, 40, 72
 - ap_poll.h, 25
 - enviro.h, 141
 - osi_lib.h, 25
- housekeeping, 13, 22

I

- include files, 25
- indication, 5
- initializing an APLI instance, 15, 31
- initiating an association, 97
- initiator, 16, 65
- instance, 9, 11, 15
 - identifier, 9

ISO

- 7489, 16
- 7498, 2
- 8326, 4
- 8327, 4
- 8649, 3
- 8650, 3
- 8822, 3
- 8823, 3

K

- kind, 43

L

- libapli.a, 25
- libapli.so.1, 25
- libraries, 25
- linking APLI applications, 25

M

- major-activity token, 21
- Makefile, 26

memory
 allocation, 18, 31
modifying the APLI environment, 15

N

negotiating presentation contexts, 16
negotiation, 6, 16
nfds, 45
non-blocking mode, 20, 22

O

objid_t, 59, 80
opening an APLI instance, 15, 29
orderly release service, 4
OSI
 abstract syntax, 4, 19
 ACSE, 3
 association control, 3
 presentation layer, 3
 reference model, 2
 service primitives, 5
 session layer, 4
 specifications, 1
 transfer syntax, 3, 19
osi_buf, 25, 37, 41, 126, 128
osi_lib.h, 25

P

P_DATA_IND, 10, 128
P_DATA_REQ, 10, 19, 126
p_data_req_cd_t, 126
P_TOKENGIVE_IND, 10, 132
p_tokengive_ind_cd_t, 132
P_TOKENGIVE_REQ, 10, 19, 130
p_tokengive_req_cd_t, 131
P_TOKENPLEASE_IND, 10, 136
p_tokenplease_ind_cd_t, 136
P_TOKENPLEASE_REQ, 10, 19, 134
p_tokenplease_req_cd_t, 135
poll(2), 22, 45

polling an APLI instance, 22, 45
presentation
 address, 16, 55, 60
 context, 16, 151
 context definition list, 16, 62
 context definition response list, 16, 63
 context identifier, 59
 context result, 16
 layer, 1, 3, 151
 services, 7, 17
primitives, 17, 89
program development summary, 12
protocol
 data unit, 19
 data units, 3, 4
 information, 10, 18, 37

R

receiving
 protocol information, 40
 service primitives, 17, 40
 user data, 41
release
 request, 118
 token, 21, 71
releasing, 43
 an association, 118
 memory, 22, 43
remote address, 65
reporting errors, 22, 47
request, 5, 37
 abnormal release, 93
 release, 95
responder, 16, 65
response, 5, 37
revents, 45
running the example application, 143
runtime libraries, 25

S

sending
 primitives using multiple calls, 20

- protocol information, 18
- service primitives, 17
- user data, 19, 126
- service
 - activity, 5
 - confirmation, 6
 - data transfer, 5
 - indication, 5
 - orderly release, 4
 - primitives, 5, 9, 10, 17, 89
 - provider, 7, 29
 - request, 5
 - response, 5
 - synchronization, 5
 - user, 7
- session layer, 4, 151
- session-connection-identifier, 56, 58
- setting
 - presentation addresses, 16, 55, 60, 65
 - the execution mode, 20
 - token assignment, 21
- sptype, 37, 40
- state tables, 81, 90
- status, 8
- SunLink OSI, 1, 9, 11, 25, 29
- synchronization, 21
 - point serial number, 60
 - service, 5
 - token, 71
- synchronize-minor token, 21

T

- terminating an APLI instance, 49
- timeout, 46
- token
 - assignment, 21
 - give up token, 130
 - receipt indication, 132
 - request indication, 136
- token request, 134
- transfer syntax, 3, 19, 151
- transport layer, 151

U

- ubuf, 37, 41
- unavailable tokens, 21
- unconfirmed negotiation, 6
- upper layer infrastructure, 2
- user data, 10, 16, 19, 37
 - encoding, 19