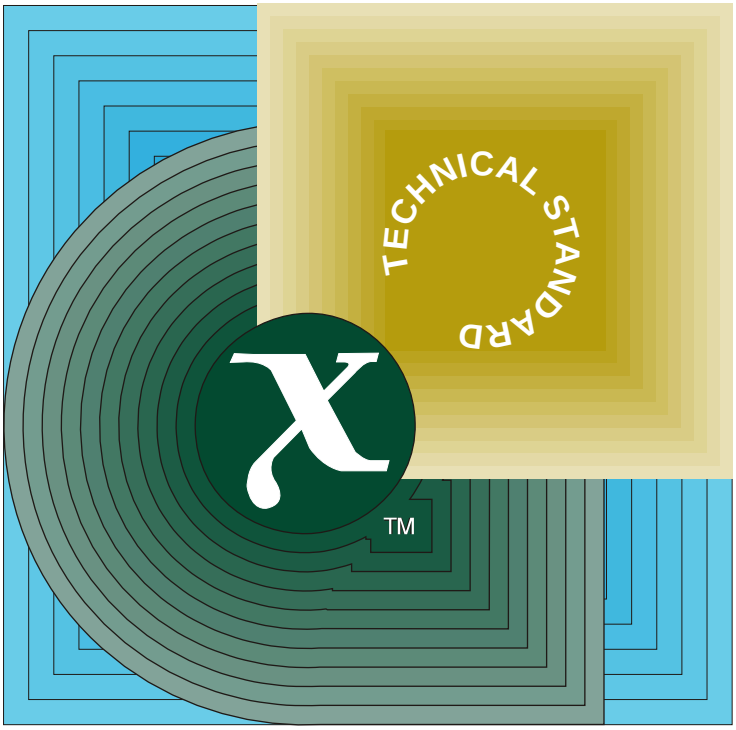
Technical Standard

ACSE/Presentation Services API (XAP)



THE Open GROUP

[This page intentionally left blank]

X/Open CAE Specification

ACSE/Presentation Services API (XAP)

X/Open Company Ltd.

X/Open CAE Specification

ACSE/Presentation Services API (XAP)

ISBN: 1-872630-91-X

X/Open Document Number: C303

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

List of Figures

List of Tables

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and allows users to move between systems with a minimum of retraining.

The components of the Common Applications Environment are defined in X/Open CAE Specifications. These contain, among other things, an evolving portfolio of practical application programming interfaces (APIs), which significantly enhance portability of application programs at the source code level, and definitions of, and references to, protocols and protocol profiles, which significantly enhance the interoperability of applications.

The X/Open CAE Specifications are supported by an extensive set of conformance tests and a distinct X/Open trademark - the XPG brand - that is licensed by X/Open and may be carried only on products that comply with the X/Open CAE Specifications.

The XPG brand, when associated with a vendor's product, communicates clearly and unambiguously to a procurer that the software bearing the brand correctly implements the corresponding X/Open CAE Specifications. Users specifying XPG-conformance in their procurements are therefore certain that the branded products they buy conform to the CAE Specifications.

X/Open is primarily concerned with the selection and adoption of standards. The policy is to use formal approved *de jure* standards, where they exist, and to adopt widely supported *de facto* standards in other cases.

Where formal standards do not exist, it is X/Open policy to work closely with standards development organisations to assist in the creation of formal standards covering the needed functions, and to make its own work freely available to such organisations. Additionally, X/Open has a commitment to align its definitions with formal approved standards.

X/Open Specifications

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) Specifications are the long-life specifications that form the basis for conformant and branded X/Open systems. They are intended to be used widely within the industry for product development and procurement purposes.

Developers who base their products on a current CAE Specification can be sure that either the current specification or an upwards-compatible version of it will be referenced by a future XPG brand (if not referenced already), and that a variety of compatible, XPG-branded systems capable of hosting their products will be available, either immediately or in the near future.

CAE Specifications are not published to coincide with the launch of a particular XPG brand, but are published as soon as they are developed. By providing access to its specifications in this way, X/Open makes it possible for products that conform to the CAE (and hence are eligible for a future XPG brand) to be developed as soon as practicable, enhancing the value of the XPG brand as a procurement aid to users.

- *Preliminary Specifications*

  These are specifications, usually addressing an emerging area of technology, and consequently not yet supported by a base of conformant product implementations, that are released in a controlled manner for the purpose of validation through practical implementation or prototyping. A Preliminary Specification is not a ''draft'' specification. Indeed, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE Specification.

  Preliminary Specifications are analogous with the ''trial-use'' standards issued by formal standards organisations, and product development teams are intended to develop products on the basis of them. However, because of the nature of the technology that a Preliminary Specification is addressing, it is untried in practice and may therefore change before being published as a CAE Specification. In such a case the CAE Specification will be made as upwards-compatible as possible with the corresponding Preliminary Specification, but complete upwards-compatibility in all cases is not guaranteed.

In addition, X/Open periodically publishes:

- *Snapshots*

  Snapshots are ''draft'' documents, which provide a mechanism for X/Open to disseminate information on its current direction and thinking to an interested audience, in advance of formal publication, with a view to soliciting feedback and comment.

  A Snapshot represents the interim results of an X/Open technical activity. Although at the time of publication X/Open intends to progress the activity towards publication of an X/Open Preliminary or CAE Specification, X/Open is a consensus organisation, and makes no commitment regarding publication.

  Similarly, a Snapshot does not represent any commitment by any X/Open member to make any specific products available.

X/Open Guides

X/Open Guides provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant.

X/Open Guides are not normative, and should not be referenced for purposes of specifying or claiming X/Open-conformance.

This Document

This document is a CAE Specification.

XAP is an Application Programming Interface to the *connection-oriented* services of the Presentation Layer of the OSI protocol stack, including access to the ACSE application service element from the Application Layer. X/Open has defined this API as an interface to support portable implementations of application-specific OSI services and non-OSI applications.

This specification describes the XAP API and defines the functions and data structures which it provides for use by applications.

Structure

- Chapter 1 is an introduction.
- Chapter 2 lists XAP functions and describes how they may be used to set up associations and transfer data.
- Chapter 3 describes the XAP environment used for transferring data and control information between the XAP and the API user.
- Chapter 4 define the functions which make up XAP.
- Chapter 5 presents manual pages for XAP commands. Specifically, it describes the XAP *ap_osic*( ) command.
- Chapter 6 provides information on the format of files used by XAP. Specifically, it describes the XAP environment file, which is used by the *ap_osic*( ) command.
- Chapter 7 presents *manual pages* for each of the primitives of the underlying OSI services to which the XAP provides access via the *ap_snd*( ) and *ap_rcv*( ) functions.
- Appendix A presents a subset of the contents of the **<xap.h>** header file.

Intended Audience

It is aimed at two groups of readers:

API Implementors
: System vendors who are implementing an OSI stack may use this specification to design an XAP-conformant interface to the stack's services, facilitating support of applications from diverse sources.

Applications Implementors
: Implementors of OSI and non-OSI applications which are to run over OSI protocol stacks may use this specification to assist in the design of applications which are portable across OSI protocol stack implementations from different system vendors. Here the term OSI applications includes both application

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — command operands, command option-arguments or variable names, for example, substitutable argument prototypes
  - — environment variables, which are also shown in capitals
  - — utility names
  - — external variables, such as *errno*
  - — functions; these are shown as follows: *name*( ). Names without parentheses are C external variables, C function family names, utility names, command operands or command option-arguments.
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
- The notation [EABCD] is used to identify a return value ABCD, including if this is an an error value.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font. Brackets shown in this font, `[ ]`, are part of the syntax and do *not* indicate optional items.

Trademarks

X/Open™ and the ‘‘X’’ device are trademarks of X Company Ltd. in the U.K. and other countries.

Referenced Documents

The following documents are referenced in this specification:

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.

Chapter 1

Introduction

This document is an X/Open CAE Specification (see Preface for a description of the types of X/Open publication). It defines the X/Open ACSE/Presentation programming interface (XAP), an interface to the OSI Association Control Service Element in the Application Layer and to the Presentation Layer of the seven-layer Open Systems Interconnection Reference Model.

1.1 Objectives

X/Open has already designed an Application Programming Interface (API) which can be used to access the OSI protocol stack at the Transport Layer. The X/Open Transport Interface (XTI) provides an interface to a wide range of underlying protocol stacks (for example OSI, TCP/IP, proprietary) and is independent of the particular implementation of the underlying protocol stack. Thus, XTI facilitates portability of applications among different protocol suites and among operating systems and protocol stack implementations.

There are a number of reasons why X/Open has decided to define a second API, to access the services provided by the upper layers of the OSI stack:

- Above the Transport Layer, the OSI layer and service elements are divided into those that provide services common to all applications (for example the Presentation Layer, ACSE, and so on), and those that are specific to a particular application or group of applications (for example FTAM, X.500, etc.). In the absence of an open API to the upper layers of the OSI protocol stack, the application implementor is faced with two choices: either each application must incorporate an implementation of the upper layers; or applications must be implemented to access the upper layers via a non-standard (perhaps proprietary) API. The former approach does have the advantage of allowing the application to *tune* the upper layer implementation by closely coupling it to the application's requirements, however it has the disadvantage that the user may have to use and manage multiple implementations of the upper layers. The latter approach risks locking the user into a particular product family. A standardised interface at the highest point of commonality shared among most Application Layer services (which is the ACSE/Presentation level) allows the user to avoid this problem by making possible separation between the application-specific and common elements of the stack via an open API.
- Further, companies which have implemented or bought OSI stacks may not wish to implement each new OSI Application Layer service as it emerges. A standard interface capable of supporting these applications (like FTAM, X.500, etc.) makes it possible for system vendors and users to purchase Application Layer products from various ISVs and add them to their existing common stack for lower layers.
- Finally, there is a requirement to implement non-OSI applications directly over an OSI protocol stack. This requirement arises either because of the lack of an OSI application service suitable for supporting the particular application, or because the application already incorporates all the Application Layer services it requires. Many non-OSI applications are currently implemented using the services of the OSI Transport Layer. However, a number of characteristics of the transport service (for example, its lack of an orderly release service) make it inappropriate for the support of certain applications without some enhancement. Again, a standardised interface to the upper layers of the OSI stack provides a suitable platform upon which to implement such applications in a portable fashion.

  This latter requirement is an important element in strategies for coexistence between OSI and other protocol suites, and for migration of networks from those protocol suites to OSI, as a number of the techniques employed rely upon implementing existing applications on top of an OSI protocol stack.

For these reasons, X/Open has identified an API to the connection-oriented services of the Presentation Layer and the ACSE application-service-element as the interface to support portable implementations of application-specific OSI services and non-OSI applications.

Figure 1-1 illustrates possible APIs to OSI services and their relationship to potential applications.

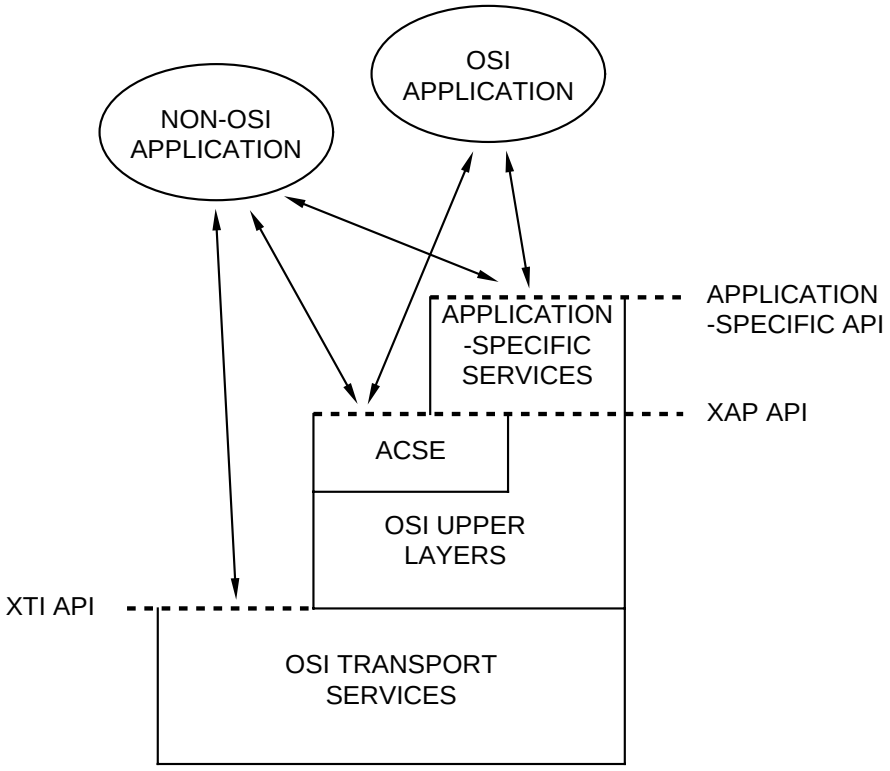


Figure 1-1 OSI Service Interfaces

Purpose

The purpose of this Specification is to describe the XAP API and to define the functions and data structures which it provides for use by applications. The specification defines the state information which controls the operation of the API and its underlying service-provider, the states in which the primitives provided by the API are valid, and the effect on the state information of each of these primitives.

It is not the purpose of this specification to define a particular subset of the ACSE and Presentation Layer protocols which implementations must support. The compliance requirements for an implementation of the API and the underlying protocol implementation to which it provides access are defined in Section 1.6 on page 10.

1.2 Audience

This specification has two specific groups of implementors as its target audience:

API Implementors
: System vendors who are implementing an OSI stack may use this specification to design an XAP-conformant interface to the stack's services, facilitating support of applications from diverse sources.

Applications Implementors
: Implementors of OSI and non-OSI applications which are to run over OSI protocol stacks may use this specification to assist in the design of applications which are portable across OSI protocol stack implementations from different system vendors. Here the term OSI applications includes both application utilities and application-specific APIs.

1.3 Structure of this Specification

- The remainder of this chapter consists of the following sections:
  - — Section 1.4 on page 6 gives a short description of the services to which the API provides access.
  - — Section 1.5 on page 9 describes terms used in the rest of the specification and describes the typographical conventions used in the *manual pages*.
  - — Section 1.6 on page 10 defines the requirements placed upon implementations of this API. These include the minimum set of functions that must be provided by an implementation of XAP, and the requirements placed upon the underlying ACSE and Presentation Layer implementations.
  - — Section 1.7 on page 11 describes areas of development of the OSI standards upon which this document is based, which may result in changes and/or extensions to the interface in a possible future version of XAP.
- Chapter 2 lists XAP functions and describes how they may be used to set up associations and transfer data. This chapter also describes the data structures provided for transferring data and control information between XAP and the API user and shows how these data structures are used.
- Chapter 3 describes the XAP environment attributes and the data structures used for transferring data and control information between the XAP and the API user.
- Chapter 4 presents the *manual pages* for the XAP API. These define the functions which make up XAP, providing the detailed specifications of parameters and data structures. There are also manual pages defining the contents of the XAP environment and the data structures used for transferring data and control information between XAP and the API user.

  The *manual pages* for *ap_snd*( ) and *ap_rcv*( ) include tables which define the valid states in which each primitive can be sent or received, the resulting state, and the effect on the variables that control the protocol's operation.
- Chapter 5 presents manual pages for XAP commands. Specifically, it describes the XAP *ap_osic*( ) command.
- Chapter 6 provides information on the format of files used by XAP. Specifically, it describes the XAP environment file, which is used by the *ap_osic*( ) command to compile a file that the *ap_init_env*( ) function can use to initialise the XAP Library environment.
- Chapter 7 presents *manual pages* for each of the primitives of the underlying OSI services to which the XAP provides access via the *ap_snd*( ) and *ap_rcv*( ) functions. Each manual page provides a short description of an ACSE or Presentation Layer primitive, including the circumstances under which it may be sent or received, and a detailed description of the parameters associated with it.
- Appendix A presents a subset of the contents of the **<xap.h>** header file.

1.4 Overview of ACSE/Presentation Services

Readers of the subsequent chapters of this specification must be familiar with the services provided by the underlying OSI protocol stack. This section provides a brief overview of the services of the upper layers of the OSI protocol stack for the benefit of those readers who require it. The text includes references to the OSI specifications for the services and protocols under discussion.

Upper Layer Architecture

Within the OSI Basic Reference Model the Transport Layer and the layers below cooperate to provide an end-to-end transmission path using the 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 as the Transport Layer provides a consistent service regardless of the variations in the layers below. These layers are known as the Upper Layers and consist of the Session Layer, Presentation Layer, and the Application Layer (which is organised internally as a set of application-service-elements). Figure 1-2 gives a simplified picture of how these layers fit together. The layers are discussed individually below.

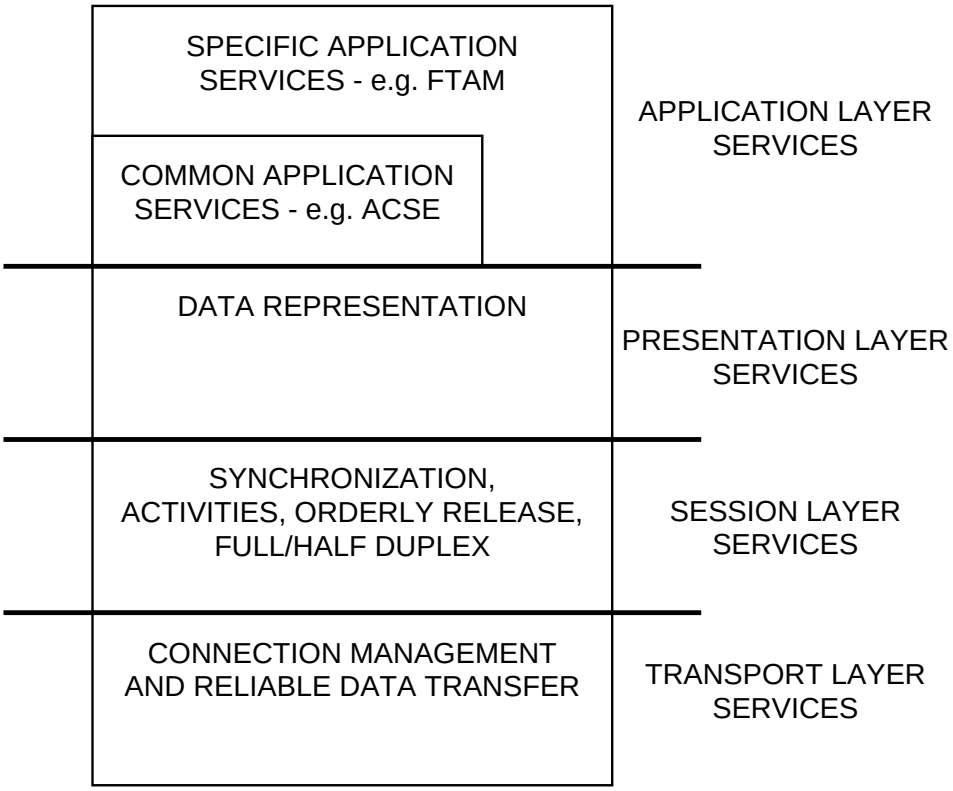


Figure 1-2 OSI Upper Layers

Application Layer

This layer provides the means by which OSI Applications access the services of an OSI protocol stack. The OSI Basic Reference Model describes communications between *application-entities* (defined as the aspects of an application process pertinent to OSI). An application entity consists of the communications aspects of the application plus a number of *application-service-elements* which are part of the Application Layer and provide OSI services to the user. Application-service-elements 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 for establishing and releasing 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 Common Management Information Protocol (CMIP), on the other hand, uses the services of the Remote Operation Service (ROSE) to exchange its PDUs.

The ACSE application-service-element (reference **ISO 8649** and **ISO 8650**) defines service primitives which map directly to the equivalent primitives of the Presentation and Session Layers 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

The Presentation Layer (reference **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 size of integers). Future extensions to the Presentation Layer may include provision of support for data compression and data encryption.

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 unit to the session layer. The peer Presentation Layer performs the opposite transformation before passing the resulting data unit to its application entity.

The layer negotiates, on behalf of the application entity, a set of *abstract syntaxes* that are to be used during the session. (An abstract syntax is defined by a protocol specification to describe the structure of the Protocol Data Units which 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 ACSE's data units), FTAM (to represent FTAM's data units) and one or more syntaxes for the FTAM document types that are to be used during this FTAM session.

Each data value passed to the Presentation Layer specifies the abstract syntax 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 which 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

The Session Layer (reference **ISO 8326** and **ISO 8327**) enhances the basic end-to-end service of the Transport Layer by providing services which allow applications to organise and synchronise their interactions and data transfers. These services are supported by a set of tokens and service primitives which control their exchange, the use of which is negotiated during session connection establishment.

In addition to connection establishment and data transfer services which map closely onto 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.
- **Synchronisation Services**.
  These services allow data transfer to be controlled for the purposes of recovery. The major sychronisation point service divides a data exchange into dialogue units, the data in each dialogue unit being confirmed to have been received correctly before the next unit can be started. The major and minor synchronisation point services identify points in the data transfer at which recovery may occur. The resynchronisation service allows the data transfer to be restarted at an identified synchronisation point (not earlier than the last major synchronisation point). The right to issue synchronisation 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 dialogue 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.

1.5 Terminology and Conventions

Definition of Terms

The terminology used in this specification is that defined in the OSI standards which define the services to which XAP provides access. For convenience, the Glossary on page 231 provides brief definitions for these terms. Terms which have specific meaning for the XAP API are defined in Section 2.1 on page 14.

Use of Naming Prefixes

In order to preserve uniqueness, all functions, typedefs, data items and constants defined by this specification have names that begin with the prefix *ap_* or AP_.

While the AP_ prefix is used on the symbolic constant which identifies a primitive, it is not applied to the primitive name itself, to avoid making primitive names unecessary unwieldy.

Alignment with ISO C

As part of updating the XAP PS to CAE specification status, the definition of the API has been modified to align it with the ISO C standard.

- The function definitions use the ISO C function declaration syntax
- The "const" type qualifier has been added to those arguments that are treated as "read-only" by the API functions.

1.6 XAP Compliance

All the XAP functions, as listed in Table 2-1 on page 17, must be provided. However, the *functionality* defined for some of these functions is *optional*. When a function call is made requesting an optional service which is not available, the implementation must return the error code [AP_NOT_SUPPORTED].

In particular, a conforming implementation may choose to provide no support for:

- the use of *ap_save*( ) and *ap_restore*( ) as a method of sharing an XAP instance between cooperating processes
- the use of *ap_look*( ) and *ap_restore*( ) with a NULL *savef* argument as a way of supporting Association Listening
- the use of *ap_look*( ) to examine incoming primitives
- the use of the *ap_ioctl*( ) function
- the use of the *ap_osic* command
- the *ap_init_env*( ) function when called with a non-null *env_file* argument
- Selector and NSAP wildcarding in the Presentation Address.

When a request for such unsupported functionality is made, the implementation must return the error code [AP_NOT_SUPPORTED].

An implementation which complies with this specification shall also comply with the Conformance clauses of the ISO/IEC protocol specifications which this specification references. These Conformance clauses specify:

- requirements on combinations of functional units
- by implication, permitted sequences of primitives.

It should be noted that the ISO/IEC Session Protocol Conformance clause is currently being revised. When this revision is accepted, the revised Conformance clause will apply.

The choice of underlying profile determines which XAP Service primitives, as listed in Table 2-2 on page 19, are supported. The features of the underlying profile are stated in the appropriate Protocol Implementation Conformance Statement (PICS). PICS Proforma for the ISO/IEC Session, Presentation and ACSE Protocol Specifications are currently under ballot. Conformant completion of these PICS Proforma shall also apply to implementations claiming conformance to this specification, once these PICs have been approved by ISO/IEC.

International Standardized Profiles (ISP) allow a reduced set of underlying features to be specified, by placing restrictions on the PICS. These restrictions are in terms of a requirements list - effectively deltas to the (protocol) PICS status column - and contain additional questions relevant to the profile. For example, an ISP may define some parameters as *out of scope*, enabling the implementation to ignore them. International Standardized Profiles for ISO/IEC Session, Presentation and ACSE are under development as ISO/IEC pDISP 11188 - for example, ‘‘Common Upper Layer Requirements - Part 3: Minimal Upper Layer Facilities’’. These are expected to reach ISP status by Q1/94.

The list of relevant ISPs is given in the XAP Component Definition. Support for specific profiles Shall be declared by the vendor in the XAP Conformance Statement Questionnaire (CSQ).

1.7 Future Directions

This specification is issued at "CAE" status. It will be updated from time to time in order to maintain alignment with the International Standards which support it or to address support issues.

Application Context Negotiation During Association Establishment

An Amendment to the ACSE Service and Protocol is currently being prepared. The proposed change introduces the Application Context Negotiation functional unit and a new parameter to the A-ASSOCIATE request/indication/response/confirm service. If the proposed change is accepted by ISO/IEC, it will be visible as:

- an addition to the values permissible in the XAP library environment attribute for AP_AFU_AVAIL and AP_AFU_SEL, to introduce the new functional unit
- an addition to the ap_a_assoc_env_t structure to introduce the application context name list
- an additional question in the XAP CSQ on support of the functional unit and related parameters.

ACSE Authentication

Amendment 1 to the ACSE Service Definition (ISO 8649) and to the ACSE Protocol Specification (ISO 8650) define authentication during association establishment. This facility provides for a two-way exchange of information that can support authentication methods including password mechanisms.

X/Open may consider updating this specification to support the new features defined therein.

Presentation Context Management and Restoration

This XAP specification has no provision for the Presentation Context Management functional unit or the Context Restoration functional unit.

X/Open may consider updating this specification to support these Presentation functional units at a later date.

Chapter 2

Overview of XAP

This chapter provides an overview of the XAP interface, describing the model upon which it is based, its functions, and the mechanisms provided for communication with it. A brief example of how the interface may be used to establish an association and transfer data is also provided.

2.1 XAP Model

Figure 2-1 shows the main features of the model upon which the XAP interface is based.

XAP provides functions for establishing associations and transferring data using ACSE and Presentation Layer service primitives, and an *environment* which is used to store information for use in the current association. Requests and responses from the *service user* are combined with information from the XAP environment and passed as ACSE or Presentation Service primitives to the *service provider*. Indications and confirmations from the service provider update the XAP environment and are passed to the service user.

The features and terminology of the XAP Model are discussed below.

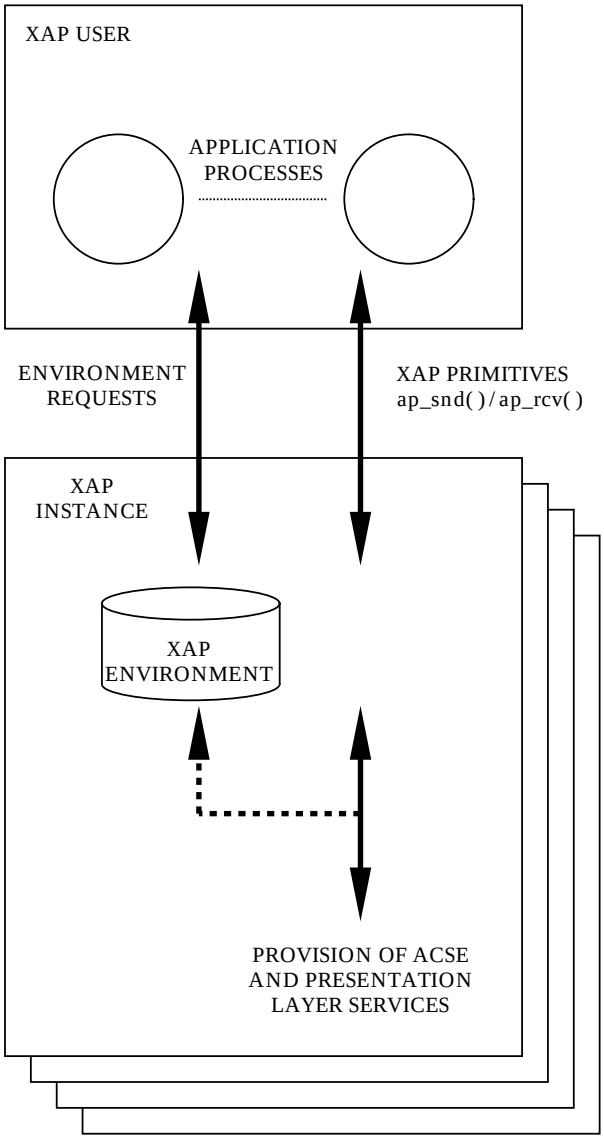


Figure 2-1 XAP Model

Service User

The XAP *service user* is an application which uses the services of the OSI ACSE and Presentation Layer services to implement its functions. This service user corresponds to the concept of an *application entity*, defined by the OSI reference model, ISO 7498. Depending on the architecture of the operating system and the requirements of the application, the service user may be a single-threaded or multi-threaded program or a group of cooperating processes.

Possible service users are OSI and non-OSI applications (for example an *FTAM initiator* utility or an application ported from another protocol suite), or a higher-level API (such as an implementation of X/Open's *X.400 API* or the non-OSI *Xlib* which provides access to the primitives of the X-Window system).

Service Provider

XAP is a standardised interface to the services of an implementation of the ACSE and Presentation layers, known as the *service provider*. An end-system may support one or more service providers via the XAP interface. A particular service provider is selected by specifying a *service provider identifier* when an XAP Instance is created. This specification does not assign any syntax to this identifier, other than that it must be a character string. Individual implementations may specify additional syntax and semantics. For example, an implementation might require that the identifier is a valid UNIX pathname, identifying a special file associated with the selected XAP service provider.

The XAP specification does not define explicitly the service options that must be supported by the provider; the requirements placed upon the local implementation are defined in Section 1.6 on page 10.

XAP Instance

An XAP *instance* is the collection of information and OSI communications capabilities required to establish and maintain an application association with another application entity invocation. The instance includes the *XAP environment* and any internal state information required by the XAP implementation. A separate XAP *instance* is created to support each association that a service user wishes to establish. The user may have several associations in progress at one time.

When the XAP *instance* is created, an XAP *instance identifier* is returned which is used to identify this instance in subsequent calls to XAP functions. (The associated XAP environment must first be initialised before the instance can be used to send and receive primitives.) This identifier is meaningful only within the context of the process which created the *instance*. XAP does not define how an XAP *instance* is passed from one process to another.

XAP Environment

The XAP environment is the repository for all of the information necessary to establish and maintain an association with another application entity. A single piece of information stored in the environment is referred to as an *attribute*. A service user can read or modify characteristics of an association or the OSI protocol stack that supports it by performing operations on these attributes.

Many of these attributes correspond to parameters associated with the ACSE service primitives. Thus the environment provides a mechanism for setting information that is to be used when establishing an association. See **Service Primitive Parameters** below for more details.

There are two types of attributes: read-write attributes and read-only attributes. Read-only attributes are those that cannot be set by the service user. They typically store either static information (for example, available protocol versions) or status information (for example,

current XAP state). All other attributes are both readable and writable by the application.

It should be noted that an attribute's classification as *read-only* or *read-write* does not imply that those operations may be performed on it at any time. To the contrary, some *read-only* attributes are readable only in certain states. Similarly, *read-write* attributes are not necessarily readable and writable in every or even the same states. Rather they are distinguished from *read-only* attributes by the fact that they are writable in some state. Complete information about the readability and writability of each attribute is provided in the description for **ENVIRONMENT** in Chapter 3.

Service Primitive Parameters

With the exception of the *user data* parameter (discussed below), the parameters passed to the service provider by XAP when sending a particular primitive are derived from attributes held in XAP's environment and from *control data* passed to XAP by the service user with each primitive request. Control data allows the service user to pass parameters specific to each primitive and to override some of the values held in the environment when establishing an association.

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

For example, when an A_ASSOC_IND primitive is received, the values of several attributes may change. One such attribute is AP_PCDL which is set to reflect the presentation context definition list specified by the application requesting the association. The effect of each XAP primitive on the environment is discussed in the manual page descriptions for *ap_snd*( ) and *ap_rcv*( ).

User Data

Almost all primitives to which XAP provides access, accept a *user data* parameter. For OSI protocols, the content of *user data* is generally 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 information* parameter). For non-OSI protocols, the content of *user data* is not constrained. Unlike other parameters, *user data* must be encoded and decoded by the service user. The descriptions of the individual primitives, given in Chapter 7, indicate how data should be encoded in each case.

2.2 XAP Functions and Mechanisms

The XAP functions can be divided into the following categories:

- functions used to establish and release an XAP instance - *ap_open*( ), *ap_close*( )
- functions used to manage the XAP environment - *ap_init_env*( ), *ap_set_env*( ), *ap_get_env*( ), *ap_free*( )
- a function used to bind a Presentation Address with an XAP instance - *ap_bind*( )
- functions used to send/receive XAP primitives - *ap_snd*( ), *ap_rcv*( ), *ap_look*( ), *ap_poll*( )
- a function used to enable or disable software interrupts for an XAP instance - *ap_ioctl*( )
- functions used to facilitate sharing an XAP instance - *ap_save*( ), *ap_restore*( )
- functions to retrieve error messages for *aperrno* error codes - *ap_error*( )

Table 2-1 lists these functions. The functions and the interface mechanisms that support them are discussed in more detail in the sections that follow.

Functions	**Parameters**
ap_open	(provider, oflags, ap_user_alloc, ap_user_dealloc, aperrno_p)
ap_close	(fd, aperrno_p)
ap_bind	(fd, aperrno_p)
ap_snd	(fd, sptype, cdata, ubuf, flags, aperrno_p)
ap_rcv	(fd, sptype, cdata, ubuf, flags, aperrno_p)
ap_save	(fd, savefd, aperrno_p)
ap_restore	(fd, savefd, oflags, ap_user_alloc, ap_user_dealloc, aperrno_p)
ap_get_env	(fd, attr, val, aperrno_p)
ap_set_env	(fd, attr, val, aperrno_p)
ap_init_env	(fd, env_file, flags, aperrno_p)
ap_ioctl	(fd, request, argument, aperrno_p)
ap_error	(aperrno)
ap_free	(fd, kind, val, aperrno_p)
ap_look	(fd, sptype, cdata, ubuf, flags, aperrno_p)
ap_poll	(fds, nfds, timeout, aperrno_p)

Table 2-1 XAP Functions

2.2.1 Establishing and Releasing an XAP Instance

In order to use XAP, the service user must first create an XAP instance. This is accomplished by using the *ap_open*( ) function. It is possible that an instance may also be obtained through some other mechanism as well. However, such mechanisms depend upon the details of a specific implementation and are thus outside the scope of this interface specification. As part of the *ap_open*( ) call, the service user identifies the particular service provider that is to support this XAP instance. XAP defines the provider identifier as an uninterpreted string. Individual implementations may assign additional semantics to this string according to local conventions. For example, a CAE-conformant operating system may require this string to be a *pathname* identifying a *special file* that is associated with the service provider. The function returns an integer that is used to identify this instance in subsequent interactions with the XAP Library.

The *ap_close*( ) function is used to indicate that the indicated instance is no longer needed and that the resources required to support it can be returned to the system.

2.2.2 Reuse of an XAP Instance

After an association is terminated (normally or abnormally) the state of the XAP instance is set to AP_IDLE. The instance can either be closed or used to establish another association. Prior to establishing another association, the service user may reset any of the environment variables (see Section 2.2.3).

2.2.3 Managing the XAP Environment

Three functions are provided to perform operations on the XAP environment.

1. *ap_init_env*( ) - allocates space for the XAP environment, if required, and initialises the attributes
2. *ap_set_env*( ) - sets a particular (writable) attribute to a specified value
3. *ap_get_env*( ) - retrieves the value of a particular (readable) attribute.

In some cases, the *ap_set_env*( ) and *ap_get_env*( ) XAP environment functions pass or return multi-level structures (for example, the **ap_cdl_t** structure, used to pass the value of the context-definition-list environment variable). To assist service users in handling such structures, XAP provides the *ap_free*( ) function which will free the memory allocated to dependent structures and character strings. The *ap_free*( ) is passed the *attribute type* as specified in the environment call that allocated the structures.

2.2.4 Sending and Receiving XAP Service Primitives

The services offered by the ACSE Presentation Layer service provider are made available to the service user through a collection of XAP service primitives that are sent and received using the *ap_snd*( ) and *ap_rcv*( ) functions. The ACSE and Presentation Layer services available through this version of the XAP Interface are shown in Table 2-2 on page 19, together with the related XAP service primitives.

Services	Service Primitives			
	Send		Receive	
A-ASSOCIATE	A_ASSOC_REQ	A_ASSOC_RSP	A_ASSOC_IND	A_ASSOC_CNF
A-RELEASE	A_RELEASE_REQ	A_RELEASE_RSP	A_RELEASE_IND	A_RELEASE_CNF
A-ABORT	A_ABORT_REQ		A_ABORT_IND	
A-P-ABORT	A_PABORT_REQ[2]		A_PABORT_IND	
P-DATA	P_DATA_REQ		P_DATA_IND	
P-CAPABILITY-DATA	P_CDATA_REQ	P_CDATA_RSP	P_CDATA_IND	P_CDATA_CNF
P-TYPED-DATA	P_TDATA_REQ		P_TDATA_IND	
P-EXPEDITED-DATA	P_XDATA_REQ		P_XDATA_IND	
P-TOKEN-GIVE	P_TOKENGIVE_REQ		P_TOKENGIVE_IND	
P-TOKEN-PLEASE	P_TOKENPLEASE_REQ		P_TOKENPLEASE_IND	
P-CONTROL-GIVE	P_CTRLGIVE_REQ		P_CTRLGIVE_IND	
P-SYNC-MINOR	P_SYNCMINOR_REQ	P_SYNCMINOR_RSP	P_SYNCMINOR_IND	P_SYNCMINOR_CNF
P-SYNC-MAJOR	P_SYNCMAJOR_REQ	P_SYNCMAJOR_RSP	P_SYNCMAJOR_IND	P_SYNCMAJOR_CNF
P-RESYNCHRONIZE	P_RESYNC_REQ	P_RESYNC_RSP	P_RESYNC_IND	P_RESYNC_CNF
P-U-EXCEPTION-REPORT	P_UXREPORT_REQ		P_UXREPORT_IND	
P-P-EXCEPTION-REPORT			P_PXREPORT_IND	
P-ACTIVITY-START	P_ACTSTART_REQ		P_ACTSTART_IND	
P-ACTIVITY-RESUME	P_ACTRESUME_REQ		P_ACTRESUME_IND	
P-ACTIVITY-END	P_ACTEND_REQ	P_ACTEND_RSP	P_ACTEND_IND	P_ACTEND_CNF
P-ACTIVITY-INTERRUPT	P_ACTINTR_REQ	P_ACTINTR_RSP	P_ACTINTR_IND	P_ACTINTR_CNF
P-ACTIVITY-DISCARD	P_ACTDISCARD_REQ	P_ACTDISCARD_RSP	P_ACTDISCARD_IND	P_ACTDISCARD_CNF

Table 2-2 XAP Service Primitives

Complete information about the effects on the XAP interface of sending and receiving the various services primitives is provided in the manual page descriptions for *ap_snd*( ) and *ap_rcv*( ) in Chapter 4, and in the manual page descriptions for the individual primitives in Chapter 7.

Services that can be initiated by the service user may be associated with either two or four service primitives, depending on whether or not the service is confirmed. Figure 2-2 illustrates

2. While there is no P-P-ABORT request service defined in ISO 8822, the A_PABORT_REQ primitive is included in XAP to allow the service user to make it appear as though the Presentation Layer aborted the association in the case where a decoding error is detected. (As discussed later, the XAP service user is responsible for encoding and decoding certain data values.)

how an initiator and a responder may use the *ap_snd*( ) and *ap_rcv*( ) functions, together with the A-ASSOCIATE service primitives, to establish an association.

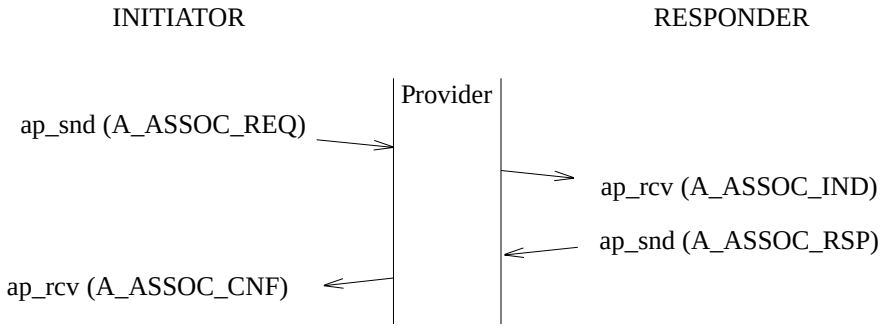


Figure 2-2 Establishing an Association

2.2.5 Sharing an XAP Instance

The *ap_save*( ) and *ap_restore*( ) functions are provided to facilitate sharing an XAP instance among cooperating processes. In some implementations, the memory allocated for storing XAP environment and state information is located in the data space of the process which created the instance. Should this process wish to pass the instance to another process, data from the address space of the first process must be copied to the address space of the second. The *ap_save*( ) and *ap_restore*( ) functions work together to accomplish this.

A common use of *ap_save*( ) and *ap_restore*( ) is to support passing of an XAP instance from a process to another, such as from a process to one of its children. Another possible use is to save default settings for a group of applications or a particular network environment, the resulting file can be used to initialise the XAP environment prior to setting attributes specific to the association being created.

The *ap_save*( ) function writes a ''snapshot'' of the storage associated with an XAP *instance* to a file that was opened by the XAP user. The file permission and file and record locking capabilities of the operating system can be used to control access to this file.

The *ap_restore*( ) function reloads the contents of an XAP snapshot into an XAP *instance*. This recreates for the referenced instance, the XAP environment and state as it was when the snapshot was created by the call to *ap_save*( ). The XAP instance into which the snapshot is restored may have been created directly by the calling user or may have been transferred from another process (the mechanism by which this transfer is achieved is outside the scope of XAP). Once the restore has been performed the XAP user can resume operations using the XAP Instance from the point where the ''saving'' process left off.

If *ap_restore*( ) is called for a transferred instance, the instance must be in the same state as it was when the snapshot was saved. No events may be sent or received in the intervening period. It should be noted that this means that *ap_save*( ) and *ap_restore*( ) cannot be used to ''roll back'' the state of the ACSE and Presentation protocol machines. Events that were processed after an XAP *instance* was saved cannot be replayed by restoring the XAP instance to its former state.

On the other hand, if *ap_save*( ) and *ap_restore*( ) are used to initialise XAP instances, the instance must be in either the AP_IDLE or AP_UNBOUND state both when the snapshot is saved and when it is restored.

2.2.6 Presentation Context Negotiation

An important aspect of association establishment is negotiation of the presentation contexts to be used during the association. When an association is to be initiated, the service user must set the AP_PCDL and AP_DPCN attributes to propose any presentation contexts that are to be used for exchanging user data (the XAP Library ensures that the presentation context used for exchange of ACSE protocol information between peer entities is available; however, the user may include it if required).

Similarly, in responding to an association request, the service user must indicate which proposed presentation contexts are acceptable by setting the AP_PCDRL and AP_DPCR attribute before issuing an A_ASSOC_RSP primitive.

It should be noted that for XAP, where all encoding and decoding of user data is performed by the XAP user, the receiving presentation service does not accept or reject any requested context. It is up to the API user to do so.

2.2.7 Presentation Addresses

As specified in the OSI Basic Reference Model (reference **ISO 7498**), an Application Entity must be addressable via a single, globally unique, Presentation Address. Thus, each XAP user has a unique Presentation Address. Multiple XAP instances created by the same XAP user may share the same Presentation Address.

An XAP user declares its Presentation Address by setting the AP_BIND_PADDR attribute and calling the *ap_bind*( ) function. This attribute must be set before any service primitives can be sent or received by the XAP user. If the XAP user is the association-initiator, it must also specify the Presentation Address of the association-responder - by setting the AP_REM_PADDR attribute - prior to issuing an A_ASSOC_REQ primitive.

An XAP library capable of simultaneously supporting more than one XAP user is capable of routing an incoming A_ASSOC_IND primitive to the particular XAP instance bound to the called Presentation Address.

XAP supports applications that are capable of receiving A_ASSOC_IND primitives addressed to any XAP user within a range of Presentation Addresses by means of *wildcard* Presentation Addresses. Two classes of wildcard addresses are recognised by an XAP implementation, *NSAP wildcard addresses* and *Selector wildcard addresses*. The two classes are discussed separately below; however XAP supports the use of both Selector wildcards and NSAP wildcards within a single presentation address. Aspects of wildcard addresses relevant to both classes are discussed in **General Comments** on page 22, while Chapter 4 includes description of how Presentation Addresses are specified.

Selector Wildcard Addresses

Different XAP users within a single computer system will likely share the same Network Address(es). Consequently, an XAP library is capable of differentiating between two unique Presentation Addresses based upon their distinct T-selector, S-selector or P-selector values, or any combination of the three. This specification does not impose any constraints on the strategy followed by an XAP implementation to achieve unambiguous Presentation Addresses.

A *Selector wildcard* Presentation Address is defined as a Presentation Address where one or more of the T-selector, S-selector and P-selector components have not been specified. The unspecified component(s) are thus allowed to match any corresponding selector value(s) specified on the called Presentation Address field of an A_ASSOC_IND primitive. It should be noted that an unspecified selector value is distinct from a null selector value.

The unspecified components of a Selector wildcard Presentation Address must follow a bottom-to-top structure. Thus, if the T-selector component is not specified, both the S-selector and the P-selector cannot be specified. Likewise, if the S-selector is not specified, the P-selector cannot be specified.

The use of *Selector wildcard* Presentation Addresses is restricted to applications which only support a responder role. They can only be used as values for the AP_BIND_PADDR attribute and can only be set when the XAP library is in the states AP_UNBOUND or AP_IDLE.

This specification does not impose any other constraints on the strategy followed by an XAP implementation to provide Selector wildcard Presentation Addresses. Some implementations may provide Selector wildcarding capabilities at multiple layers, while others may not provide them at any layer.

An XAP user bound to a Selector wildcard Presentation Address is effectively bound to the lowest level SAP(s) for which an address has been specified. For example, an XAP user bound to a wildcard address for which a T-selector value has been specified and the S- and P-selectors have been omitted is in fact bound to the Transport Layer SAP(s) identified by the specified Transport Address (consisting of the T-selector value and Network Address(es)).

The semantics of this Selector wildcard Presentation Address are that any incoming A_ASSOC_IND primitive which specifies that Transport Address will be sent to the XAP user bound to the wildcard address irrespective of the called S- and P-selector values. The XAP user is then responsible for validating the called S- and P-selector values and determining whether or not to accept the application association.

Note that multiple Selector wildcard addresses can freely coexist with each other and with fully specified addresses. XAP will always pass an incoming A_ASSOC_IND to the XAP user bound to the more specific Presentation Address that matches the called Presentation Address.

NSAP Wildcard Addresses

Where a single computer system supports multiple network addresses, XAP allows an application to listen for connection indications on one or more of these addresses by specifying a list of *NSAPs* as part of the Presentation Address assigned to AP_BIND_PADDR. Some implementations of XAP may allow an application to specify an empty list to accept connections on *any* of the local *NSAPs*; this is termed an *NSAP wildcard* address.

NSAP wildcard addresses may be used by XAP instances supporting an initiator role, a responder role, or both. In the case where an XAP instance which is bound to an NSAP wildcard address issues an A_ASSOC_REQ primitive, the service provider determines which local NSAP to use to connect to the presentation address specified in AP_REM_PADDR.

General Comments

When an indication is received by an XAP Instance which is bound to a wildcard address, the environment variable AP_LCL_PADDR is set to the specific presentation address to which the association is directed (the *called presentation address* from the A_ASSOC_REQ primitive). The address in AP_LCL_PADDR is always used as the responding presentation address when sending an A_ASSOC_RSP primitive. The XAP user may respond to an incoming A_ASSOC_IND on an address other that the called presentation address by supplying a fully specified address in AP_BIND_PADDR and calling the *ap_bind*( ) function, which in turn sets the value of AP_LCL_PADDR. It should be noted that new values may be specified for the P-selector and S-selector only. An attempt to specify values for the T-selector or *NSAP* that differ from those received in AP_LCL_PADDR results in the primitive being rejected by XAP with an error code of AP_ACCESS.

2.2.8 Memory Management Mechanisms

The *service user* may elect to handle all allocation and deallocation of memory for use by XAP (this includes memory for XAP environment attributes returned from *ap_get_env*( ), and buffers for receiving incoming data). This mechanism enables the service user to control the amount of buffer space used by XAP for receiving incoming data.

To do this, the service user specifies the entry points for allocation and deallocation routines when *ap_open*( ) is called. The service user provides two functions, *ap_user_alloc*( ) and *ap_user_dealloc*( ). The XAP implementation will call the user supplied allocation routine in the following circumstances:

- If the user has not set the AP_BUFFERS_ONLY flag on *ap_open*( ) then, when data structures are to be returned to the user by *ap_get_env*( ) or *ap_rcv*( ), when *cdata→env* is returned or *cdata→old_conn_id* is returned, the allocation function will be called with the type argument set to AP_MEMORY.
- When user data is to be returned from *ap_rcv*( ) and the user has not provided sufficient buffers and the AP_ALLOC flag was set on the call, then the allocation function will be called with the type argument set to AP_BUFFERS.

The XAP library will call the user supplied deallocation function from the *ap_free*( ) function when the user passes any structures which were allocated in the above circumstances.

XAP is responsible for freeing memory that it allocated when it is no longer required. Exceptions to this are:

- memory allocations passed to the service user as part of the *cdata* structure are described in Section 2.2.9
- buffers passed to the user containing received data
- memory allocations passed to the user on return from *ap_get_env*( ).

2.2.9 Control Data Structure

The control data structure, *cdata*, is used to pass the parameters associated with individual primitives between XAP and the service user in the *ap_snd*( ) and *ap_rcv*( ) functions. XAP maintains the parameters associated with the A_ASSOCIATE service primitives in the XAP environment. *cdata* allows the environment values to be overridden when *ap_snd*( ) is used to send an A_ASSOCIATE primitive. For *ap_rcv*( ), the structure can be used to return some environment values directly, rather than requesting them using the *ap_get_env*( ) function.

The *cdata* structure is a collation of all possible parameters to all the primitives supported by XAP. The use of *cdata* depends upon the parameters defined for the specific service primitive being sent or received. Consequently, a description of how the *cdata* parameter is used for each of the service primitives is included in Chapter 7.

When a *cdata* structure is returned by *ap_rcv*( ), XAP may have allocated data for sub-structures that are present. Once the service user has processed the data in the structure, the sub-structures may be freed by passing the *cdata* structure to *ap_free*( ).

2.2.10 Error Reporting

XAP functions return a result code as the function value. This code is zero if the function was completely successful, and –1 if any sort of error or warning condition occurred. In addition, each XAP function includes a pointer to an error code return location, *aperrno_p*, as its final argument. The service user must pass a pointer to a location into which an error code is returned if the result of the function is –1. The value in this location is unchanged if the function result is zero (that is ''SUCCESS'').

Through *aperrno_p*, XAP reports internal error conditions caused, for example, by an invalid argument or a primitive issued out of state. System errors which occur outside the scope of the XAP interface result in the location pointed to by *aperrno_p* being set to the value of the system error. In addition, several error *classes* have been identified to allow errors reported by an underlying service provider to be passed to the service user. To facilitate application portability, all implementations of XAP should adhere to this scheme for reporting errors. However, only the errors belonging to the XAP class must be supported. Specific errors belonging to the other error classes are dependent upon the underlying service providers utilised in a specific implementation. A detailed list of errors reported through *aperrno_p* is included in the introduction to the XAP interface functions, in Section 4.1 on page 56.

An *ap_error*( ) function (see *ap_error*( ) on page 63) is provided in XAP which returns a pointer to the location of a message that describes the error code passed to it.

2.2.11 Execution Mode

The XAP *ap_snd*( ) and *ap_rcv*( ) functions may be used in either *blocking* or *non-blocking* execution mode.

In blocking mode, *ap_snd*( ) blocks until resources are available to send the specified primitive in its entirety. Thus, if *ap_snd*( ) is invoked when the communication path is flow controlled, the call blocks until the entire message has been sent. Similarly, in blocking mode, *ap_rcv*( ) blocks until either an entire primitive is received, or XAP fills the buffer supplied as the *ubuf* argument. In the latter case, the AP_MORE bit of the *flags* argument is set when *ap_rcv*( ) returns. In order to receive the remainder of the primitive, the service user must continue calling *ap_rcv*( ) until the function returns with the AP_MORE bit reset.

When XAP is used in non-blocking mode, *ap_snd*( ) and *ap_rcv*( ) never block. Hence, if *ap_snd*( ) is called when insufficient resources are available to send the specified primitive in its entirety, XAP sends as much of the primitive as possible before returning with the [AP_AGAIN] error. To complete sending the primitive, the service user calls *ap_snd*( ) again with the same set of buffers as arguments until it returns successfully. A function, *ap_poll*( ), is provided which can be used to wait until resources are available to send data on a particular XAP Instance.

In the non-blocking mode, *ap_rcv*( ) reads data from the instance until an entire primitive is received, or the buffer supplied as the *ubuf* argument is full, or no more data is available to be read. Once one of these three events occurs, *ap_rcv*( ) returns. If the call returns because no more data is available, the [AP_AGAIN] error is signalled. When this error is indicated, the service user may examine the *flags* argument to determine whether a primitive was partially received. If the AP_MORE bit of the *flags* argument is set, a primitive has been partially received. The *ap_poll*( ) function can be used to wait until more data is available to be read from the instance.

The service user can request either blocking or non-blocking execution through the *oflags* argument to the *ap_open*( ) function, or by setting or resetting the AP_NDELAY bit of the AP_FLAGS environment attribute.

2.2.12 User Data Mechanisms

User Data Buffering

Most of the primitives supported by XAP include a *user data* parameter. The amount of data associated with a primitive varies from a few octets for P-EXPEDITED-DATA, to unlimited for a P-DATA primitive. In order to provide a consistent interface, a single mechanism has been designed to cater for both extremes. The mechanism separates the functions of buffer allocation and use by defining two structures: **ap_osi_dbuf_t** is a structure that controls an allocation of buffer space, containing a pointer to the start and end of the buffer (*db_base* and *db_limit*) and a reference count (*db_ref*); **ap_osi_vbuf_t** is a structure that controls the use of space, containing a pointer to the **ap_osi_dbuf_t** structure that controls a buffer (*b_datap*), and read and write pointers (*b_rptr* and *b_wptr*) which indicate the portion of that buffer reserved by this structure. It also contains a *b_cont* element, which can be used to chain a series of **ap_osi_vbuf_t** structures together.

This mechanism allows data to be passed between XAP and the service user in one or more discrete buffers, using start and end pointers to identify the portion(s) of each buffer which contains valid data. As discussed above in Section 2.2.8 on page 23, the service user may elect to provide allocation functions for the buffers which XAP uses to receive data.

Figure 2-3 shows the **ap_osi_vbuf_t** and **ap_osi_dbuf_t** structures.

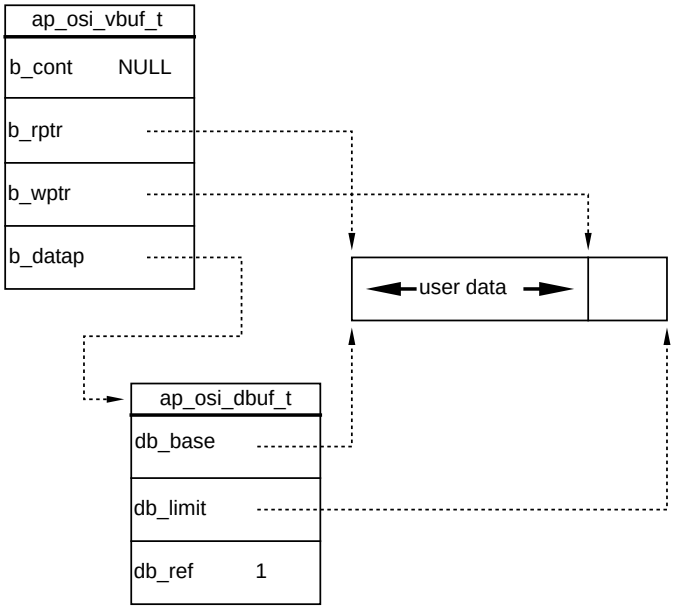


Figure 2-3 Basic Buffer Structures

Figure 2-4 shows a more sophisticated example of the use of the **ap_osi_vbuf_t** and **ap_osi_dbuf_t** structures - two separate segments of data passed in a single buffer.

For data being sent, the service user is responsible for freeing buffers once the *ap_snd*( ) function has completed. For data received, XAP will allocate the necessary buffers using the allocation functions described in Section 2.2.8 on page 23 if XAP is configured to do so and the user requests it. Again, the service user must free these buffers once the data contained in them has been processed.

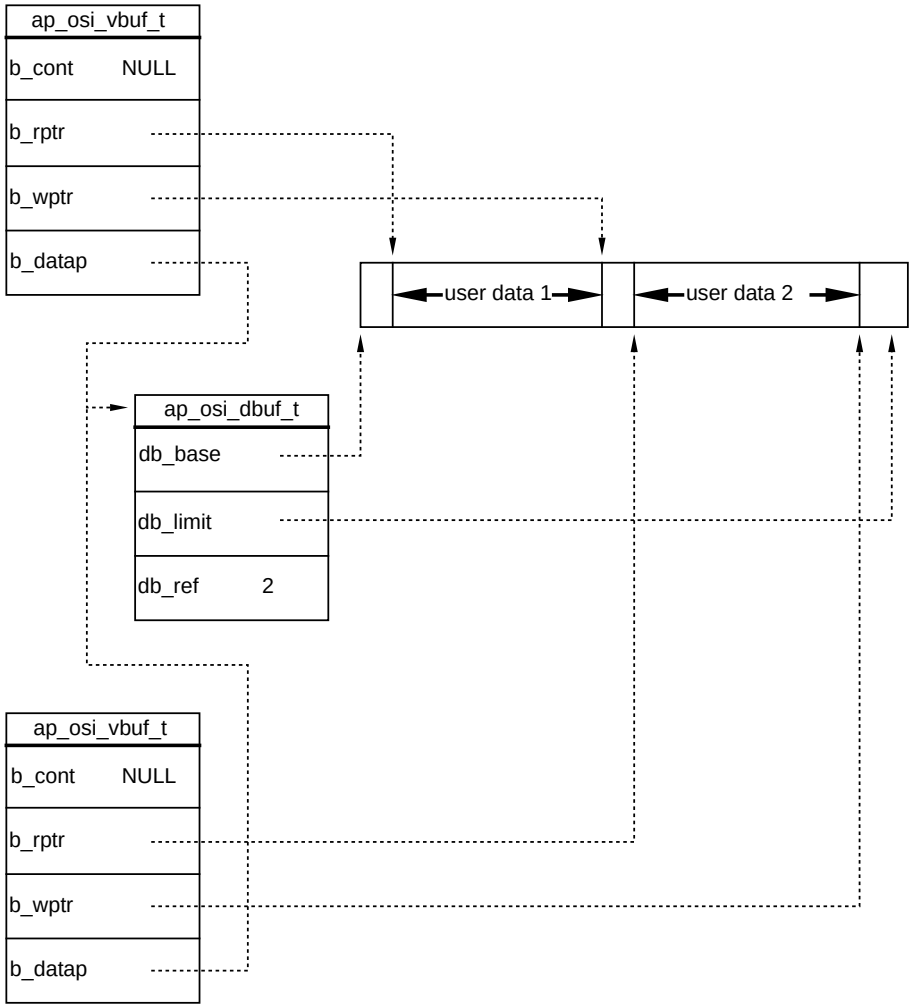


Figure 2-4 Advanced Buffer Example

Data Encoding

All *user data* passed between XAP and the XAP user is in encoded form, using an appropriate transfer syntax (see Section 2.2.6 on page 21). Thus the service user is responsible for all user data encoding and decoding.

In addition, the XAP user is responsible for generating the encoding which the ACSE and Presentation protocols require for including the user data in their PDUs. (Conceptually, this is the task of the service provider; however, XAP requires the service user to do it for the sake of efficiency.) The manual pages for the *ap_snd*( ) and *ap_rcv*( ) functions in Chapter 4, along with the manual pages for individual primitives in Chapter 7, describe how the user data is to be encoded for inclusion in a particular primitive's protocol data unit (PDU).

The application is required to encode/decode all ACSE and Presentation PDU's whenever it directly uses ACSE/Presentation primitives. When the application uses a primitive provided by a service element which maps the primitive onto ACSE/Presentation primitives, that service element shall encode/decode ACSE and Presentation PDU's. Refer to the appropriate service element API for specific information.

Two examples are given here of how information to be passed to the API must be encoded. The examples make some assumptions about how the transfer syntax for the encoding is identified and about the choices which must be made when information is passed to the ASCE and Presentation layers. The user is referred to the referenced specifications for the ACSE (ISO 8649 and ISO 8650) and Presentation (ISO 8822 and ISO 8823) protocols and for ASN.1 (ISO / IEC 8824 and ISO / IEC 8825) for precise information on this subject.

The examples use the *value syntax* defined by the ASN.1 specification with the addition (for the purposes of this discussion only) of explicit tag information included within square brackets (for example: [INTEGER] - a *universal* tag, [0] - a *context-specific* tag, and so on).

Example 1:

An FTAM initiator implementation wishes to use the ACSE protocol to set up an application association and transfer the F-CONNECT request primitive. It must encode the following data and tags into an *ap_osi_vbuf_t* structure and pass it to *ap_snd*( ) for inclusion in an A-ASSOCIATE request primitive.

```
user-information [30] {
    [EXTERNAL] {                    -- Association-information
       direct-reference [OBJECT IDENTIFIER] { ... ftam-pci(1) }
                        -- object id for the FTAM PDU abstract syntax
       single-ASN1-type [0] {
           f-initialise-request [0] {
              protocol-version    [0] { version-1 } ,
              implementation-info [1] "ACME FTAM Initiator" ,
                   .
                   .
                   .
}   }   }   }
```

Example 2:

The FTAM initiator subsequently wishes to transfer a series of strings from an unstructured text file. It must encode the following data and tags into an *ap_osi_vbuf_t* structure and pass it to *ap_snd*( ) for inclusion in a P-DATA request primitive.

```
[APPLICATION 1] {            -- fully encoded user data
       [SEQUENCE] {   -- first PDV-List
           [INTEGER] <n> ,   -- Identifies the abstract syntax
                             -- "FTAM unstructured text"
                             -- in the defined context set
           single-ASN1-type [0] {
              [GraphicString] "To begin at the beginning:"
                                  -- first string from file
       }   } ,
       [SEQUENCE] {   -- second PDV-List
           [INTEGER] <n> ,   -- Identifies the abstract syntax
                             -- "FTAM unstructured text"
                             -- in the defined context set
           single-ASN1-type [0] {
              [GraphicString] "It is spring, moonless night in the
                                 small town, starless and Bible-black..."
                                  -- second string from file
       }   } ,
                  .
                  .
}
```

Example 3:

The following encoding is an alternative to that shown in example 2.

```
[APPLICATION 1] {            -- fully encoded user data
       [SEQUENCE] {   -- first PDV-List
           [INTEGER] <n> ,   -- Identifies the abstract syntax
                             -- "FTAM unstructured text"
                             -- in the defined context set
           octet-aligned [1] {
              [GraphicString] "To begin at the beginning:"
                                  -- first string from file
              [GraphicString] "It is spring, moonless night in the
                                 small town, starless and bible-black..."
                                  -- second string from file
                  .
                  .
}   }   }
```

More Data Flag

The service user may choose not to send all of the encoded data associated with a particular primitive in a single *ap_snd*( ) call. If that is the case, the XAP Library must be informed that the primitive will be issued as a series of *ap_snd*( ) calls. This is accomplished by setting the AP_MORE bit in the *flags* parameter of the *ap_snd*( ) function. The user may continue sending data associated with the current primitive by repeatedly calling *ap_snd*( ) with the AP_MORE bit set. The *sptype* argument must remain the same for each of these calls. When the last buffer of data associated with the primitive is sent, *ap_snd*( ) must be invoked with the AP_MORE bit reset.

The *cdata* argument also should not change when a sequence of *ap_snd*( ) invocations is used to issue a single XAP primitive. However, since this argument is only examined on the first *ap_snd*( ) call in such a sequence, no error is reported if its value is subsequently modified.

Primitives that are not associated with either an ACSE or Presentation Layer PDU (for example, P_DATA_REQ, P_SYNCMINOR_REQ, P_SYNCMINOR_RSP, P_TOKENGIVE_REQ, P_TOKENPLEASE_REQ, etc.) cannot be terminated by a final *ap_snd*( ) invocation that does not carry any user data. Thus the final *ap_snd*( ) invocation always carries 1 or more octets of user data for these primitives.

If all data associated with the primitive is present on the *ap_snd*( ) call, the function must be issued with the AP_MORE bit not set.

In a similar fashion, if XAP elects to return data associated with a service primitive via multiple *ap_rcv*( ) invocations, it sets the AP_MORE bit of the *flags* argument when *ap_rcv*( ) returns. The *ap_rcv*( ) function returns with the AP_MORE bit reset when the received primitive is complete. Note that it is possible that XAP may receive a zero-length final fragment.

It should be noted that the value of the *sptype* argument must be checked after each *ap_rcv*( ) call. This is because another primitive (for example, A_PABORT_IND) could arrive before all data associated with the first primitive is processed.

2.3 Using the XAP Interface

Below is a summary of the steps required to establish an association with a remote application entity using XAP. 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.

Obtain an XAP Instance.

First, a XAP Instance is created. This is accomplished either by using *ap_open*( ), or by acquiring an already established instance through some other implementation-specific mechanism.

Initialise the XAP Environment.

Next, the XAP Environment is initialised (or restored) using the *ap_init_env*( ) (*ap_restore*( )) function.

After the environment is initialised, the user may examine or alter the environment attributes, subject to the readability and writability restrictions discussed in Chapter 3.

Bind the XAP Instance to a Presentation Address.

Before any service primitives can be sent or received, the XAP instance must be bound to a presentation address. This is accomplished by setting the AP_BIND_PADDR attribute to a value which the service user is authorised to use. The AP_BIND_PADDR attribute can be set using either the *ap_set_env*( ) function or the *ap_init_env*( ) function. Then the service provider is signalled to activate the XAP instance by using the *ap_bind*( ) function.

Set Other Environment Attributes.

In addition to AP_BIND_PADDR, other environment variables must be set before an association is established - particularly if the service user is the association-initiator. For example, before issuing an A_ASSOC_REQ primitive, the service user must specify the address of the remote entity by setting the AP_REM_PADDR attribute and set the AP_PCDL attribute to propose a list of presentation contexts that are to be used for transferring user data.

Refer to the manual pages in Chapter 4, and primitives descriptions in Chapter 7, for further information about the XAP environment attributes and how they relate to sending and receiving individual primitives.

Send or Receive A_ASSOC Service Primitives.

Once the XAP environment has been properly initialised, the service user may attempt to establish an association with a remote application entity by using the A_ASSOC service primitives in conjunction with the *ap_snd*( ) and *ap_rcv*( ) functions. Each of the A_ASSOC primitives is described in detail in Chapter 7.

Transfer Data.

Once the association is established, information may be exchanged with the remote application entity by sending and receiving the appropriate XAP primitives (using *ap_snd*( ) and *ap_rcv*( )).

Releasing the Association.

An association may be released either normally (using the A_RELEASE primitives), or abnormally (by using either the A-ABORT or the A-PABORT primitives).

2.4 Association Listeners

An Association Listener is an application which examines incoming A_ASSOC_IND primitives and passes the XAP instance to a receiving application which is to handle the incoming association.

The support for association listeners is optional; depending on the application requirements and the optional XAP functions available, the association listeners may be implemented in various ways. Implementations which support association listening are encouraged to support it in one of the following ways.

- A simple listener mechanism using no XAP optional functions:

  A listener application creates an XAP instance using *ap_open*( ), binds to an address and waits for an incoming A_ASSOC_IND. After receiving A_ASSOC_IND, the listener brings out an appropriate responding application to proceed with the association establishment. The responding application may be made available by spawning a child process or, on those systems where this is not possible, by selecting one of a pool of established reponding processes. The responding application receives the *fd* and inherits the listener's XAP environment and A_ASSOC_IND primitive and possibly data in a system dependent manner. The listener can then close the XAP instance by calling *ap_close*( ) to release local resources.

- A listener mechanism making use of the optional XAP functions *ap_look* and *ap_restore*:

  A listening application creates an XAP instance using *ap_open*( ), binds to an address and waits for an incoming A_ASSOC_IND. It calls *ap_look*( ) to obtain the A_ASSOC_IND details which it uses to select a receiving application. The listener passes the *fd* in a system dependent manner, with the pending A_ASSOC_IND primitive, to the receiving application, and closes the XAP instance by calling *ap_close*( ) to release local resources.

A receiving application calls *ap_restore*( ), with a NULL file argument to:

- create an XAP environment
- retrieve the local Presentation address to which the A_ASSOC_IND primitive was addressed
- set the XAP instance state to AP_IDLE.

The receiving application is now in a state where it can invoke *ap_rcv*( ) to receive the pending A_ASSOC_IND primitive.

Chapter 3

Environment

This chapter presents the XAP environment. It describes the XAP environment attributes and the data structures used for transferring data and control information between the XAP and the API user.

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

Below is a description of the attributes in the XAP environment.

AP_ACSE_AVAIL

The AP_ACSE_AVAIL attribute indicates which versions of the ACSE protocol are currently available.

This attribute is not used in the ''X.410-1984'' mode. Refer to the description of the AP_MODE_SEL attribute.

AP_ACSE_SEL

The AP_ACSE_SEL attribute indicates which version of the ACSE protocol has been selected for use with the current association.

This attribute is not used in the ''X.410-1984'' mode. Refer to the description of the AP_MODE_SEL attribute.

AP_AFU_AVAIL

The AP_AFU_AVAIL attribute indicates which optional ACSE functional units are currently available.

This attribute is not used in the ''X.410-1984'' mode. Refer to the AP_MODE_SEL attribute.

AP_AFU_SEL

The AP_AFU_SEL attribute indicates which optional ACSE functional units have been requested for use over the current association.

This attribute is not used in the ''X.410-1984'' mode. Refer to the AP_MODE_SEL attribute.

AP_BIND_PADDR

The AP_BIND_PADDR attribute is used to indicate the address to which this instance of XAP will be bound when *ap_bind*( ) is called.

Setting this attribute will result in the attribute AP_LCL_PADDR being set to the same value. Nevertheless, the attributes AP_BIND_PADDR and AP_LCL_PADDR do not always have the same value. For example, a *listener* application might set the AP_BIND_PADDR to a *wildcard* address and use the called address (conveyed by the AP_LCL_PADDR attribute) specified in each indication primitive to dispatch associations.

AP_CLD_AEID

The AP_CLD_AEID is the called AE invocation identifier parameter of the AARQ APDU.

This attribute is not used in the ''X.410-1984'' mode. Refer to the description of the AP_MODE_SEL attribute.

AP_CLD_AEQ

The AP_CLD_AEQ is the called AE-qualifier parameter of the AARQ APDU.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_CLD_APID

The AP_CLD_APID is the called AP invocation identifier parameter of the AARQ APDU.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_CLD_APT

The AP_CLD_APT is the called AP-title parameter of the AARQ APDU.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_CLD_CONN_ID

The AP_CLD_CONN_ID attribute conveys the value of the session connection identifier that was proposed by the association-responder.

AP_CLG_AEID

The AP_CLG_AEID is the calling AE invocation identifier parameter of the AARQ APDU.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_CLG_AEQ

The AP_CLG_AEQ is the calling AE-qualifier parameter of the AARQ APDU.

See the description of the **ap_aeq_t** data type for information about how its absence is indicated.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_CLG_APID

The AP_CLG_APID is the calling AP invocation identifier parameter of the AARQ APDU.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_CLG_APT

The AP_CLG_APT is the calling AP-title parameter of the AARQ APDU.

See the description of the **ap_apt_t** data type for information about how its absence is indicated. This attribute is not used in the ‘‘X.410-1984’’ mode.

AP_CLG_CONN_ID

The AP_CLG_CONN_ID attribute conveys the value of the session connection identifier that was proposed by the association-initiator.

AP_CNTX_NAME

The AP_CNTX_NAME attribute is the application-context-name parameter of the AARQ and AARE APDUs.

See the discussion of the **ap_objid_t** data type for information about how the absence of this parameter is indicated.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_COPYENV

The AP_COPYENV attribute is used to indicate whether certain environment attributes that correspond to parameters on the A-ASSOCIATE indication and confirmation services should be returned to the user in the *cdata* argument of the *ap_rcv*( ) function. When the value of this attribute is TRUE, these attributes are returned via the *cdata* argument.

AP_DCS

The AP_DCS attribute is the defined context set. The defined context set 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.

The value of AP_DCS shall be empty (size field of 0) in the states AP_IDLE and AP_WASSOCcnf_ASSOCreq, it shall contain the ACSE context negotiated by XAP in the state AP_WASSOCrsp_ASSOCind, and it shall contain the fully negotiated defined context set in all other states.

Note that the defined context set contains both the presentation contexts that were negotiated by the user (using AP_PCDL and AP_PCDRL) and any additional presentation contexts that were negotiated by XAP or provider. For example, the ACSE context (i.e., {joint-iso-ccitt 2 1 0 1}) will be negotiated automatically and will appear in the defined context set after negotiation is complete.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_DIAGNOSTIC

The AP_DIAGNOSTIC attribute is used to pass diagnostic details on error conditions reported by ACSE, Presentation, Session and Transport providers, in addition to that conveyed with the A_PABORT_IND primitive. This mechanism is used to provide descriptive text for all errors reported by XAP, and may be used to provide additional diagnostic information (for example, errors in incoming APDUs reported by the local ACPM, or the reason for a transport layer disconnect).

It should be noted that the information provided by this attribute is implementation-dependent - that is, an implementation may not report all of the source/reason/event combinations defined for this attribute. However, if an implementation does use the attribute to report diagnostic information, it must use the classifications defined in this specification.

AP_DPCN

The AP_DPCN attribute is the default-context-name parameter of the CP PPDU.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_DPCR

The AP_DPCR attribute is 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 value, AP_DPCR_NOVAL, indicates that it is not present.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_FLAGS

The AP_FLAGS attribute is used to control the characteristics of this instance of XAP. The currently defined flags are described below.

If the AP_NDELAY bit is set, XAP will be set to operate in non-blocking execution mode (that is, when the XAP instance is opened with the AP_NDELAY flag set, the AP_FLAG attribute is set to AP_NDELAY.)

AP_INIT_SYNC_PT

The AP_INIT_SYNC_PT attribute conveys the desired initial session synchronisation point serial number.

AP_INIT_TOKENS

The AP_INIT_TOKENS attribute conveys the specified initial assignment of tokens. The values which may be set by the initialising user, and by the responding user, are specified in Section 4.1.4 on page 60.

AP_LCL_PADDR

The AP_LCL_PADDR attribute holds the value used as the presentation address of the local entity. If the local entity is the initiator of an association, this value will be used as the calling presentation address. If the local entity is the association-responder, this value will be used as the called presentation address when the A_ASSOC_IND primitive is received and as the responding presentation address when the A_ASSOC_RSP primitive is issued.

This attribute is not directly settable by the user. However, setting the AP_BIND_PADDR attribute will result in the AP_LCL_PADDR attribute being set to the same value. Nevertheless, the attributes AP_BIND_PADDR and AP_LCL_PADDR do not always have the same value. For example, a *listener* application might set the AP_BIND_PADDR to a *wildcard* address and use the called address (conveyed by the AP_LCL_PADDR attribute) specified in each indication primitive to dispatch associations.

AP_LIB_AVAIL

The AP_LIB_AVAIL attribute indicates which versions of XAP are available to the user.

AP_LIB_SEL

The AP_LIB_SEL attribute is used to indicate which version of XAP is used. This attribute must be set before any other attributes are set or any primitives are issued.

AP_MODE_AVAIL

The AP_MODE_AVAIL attribute indicates the available modes of operation for XAP and Provider. The modes that may be supported are *normal* (AP_NORMAL_MODE) and *X.410-1984* (AP_X410_MODE).

AP_MODE_SEL

The AP_MODE_SEL attribute indicates the mode (‘‘normal’’ or ‘‘X.410-1984’’) in which XAP and Provider are to be used. This value is used as the mode parameter of the CP PPDU.

When this attribute is set to AP_X410_MODE (indicating the ‘‘X.410-1984’’ mode is in effect), certain other attributes are not used. The readability and writability of these attributes is not affected; but their values will not be used or updated by XAP. A list of the attributes that are not used in the ‘‘X.410-1984’’ mode is provided below.

Attributes Not Used in X.410-1984 Mode			
AP_ACSE_AVAIL	AP_CLD_APT	AP_DPCN	AP_PRES_SEL
AP_ACSE_SEL	AP_CLG_AEID	AP_DPCR	AP_RSP_AEID
AP_AFU_AVAIL	AP_CLG_AEQ	AP_PCDL	AP_RSP_AEQ
AP_AFU_SEL	AP_CLG_APID	AP_PCDRL	AP_RSP_APID
AP_CLD_AEID	AP_CLG_APT	AP_PFU_AVAIL	AP_RSP_APT
AP_CLD_AEQ	AP_CNTX_NAME	AP_PFU_SEL	
AP_CLD_APID	AP_DCS	AP_PRES_AVAIL	

AP_MSTATE

If the XAP 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) then AP_RCVMORE bit is set. (Note that it is possible for both bits to be set.)

AP_OLD_CONN_ID

The AP_OLD_CONN_ID attribute conveys the values of the session connection identifiers from a previous session.

AP_OPT_AVAIL

The AP_OPT_AVAIL attribute is used to indicate what optional functionality is supported in the underlying protocol implementation.

AP_XXXX_WILD indicates whether address wildcarding is supported at the specified level. (XXXX can be NSAP, TSEL, SSEL or PSEL.)

AP_PCDL

The AP_PCDL attribute is used to propose the list of presentation contexts to be used on the connection. The association-initiator sets this list to indicate the contexts it wishes to use for encoding user data on the association. Since the 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, XAP 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).

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_PCDRL

The AP_PCDRL attribute is used to indicate 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 XAP, and appears in the value of AP_DCS.

This attribute is not used in the ''X.410-1984'' mode. Refer to the description of the AP_MODE_SEL attribute.

AP_PFU_AVAIL

The AP_PFU_AVAIL attribute indicates which optional Presentation functional units are currently available.

This attribute is not used in the ''X.410-1984'' mode. Refer to the description of the AP_MODE_SEL attribute.

AP_PFU_SEL

The AP_PFU_SEL attribute indicates which optional Presentation Layer functional units have been requested for use over the current association.

This attribute is not used in the ''X.410-1984'' mode. Refer to the description of the AP_MODE_SEL attribute.

AP_PRES_AVAIL

The AP_PRES_AVAIL attribute indicates which versions of the Presentation Layer protocol are currently available. This attribute is not used in the ''X.410-1984'' mode.

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.

This attribute is not used in the ''X.410-1984'' mode. Refer to the description of the AP_MODE_SEL attribute.

AP_QLEN

The AP_QLEN attribute is used to indicate the number of connection indications that will be held by the provider when all of the connection endpoints bound for listening at AP_BIND_PADDR are in use. The user can set this attribute to request a specific queue size. The provider may change the value if it cannot support the requested queue size or if another process bound to the same address has already requested a larger queue. The user can determine the current queue size by examining the value of AP_QLEN using *ap_get_env*( ).

AP_QOS

The AP_QOS attribute is used to specify the quality of service requirements for the association. It holds the ranges of QOS values that the XAP user is willing to accept.

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.

AP_ROLE_ALLOWED

The AP_ROLE_ALLOWED parameter is used to specify how an instance of XAP may be used. Specifically, the value of this attribute indicates whether the XAP instance may be used to send an A_ASSOC_REQ primitive, receive an A_ASSOC_IND primitive, or both. Note that this attribute only affects the roles that an instance of XAP may play with respect to association establishment. Changing the value of this attribute will not affect the way in which XAP participates in an association that has already been established.

It should be noted that an A_ASSOC_IND primitive may be received even after this attribute has been set to prohibit receipt of association indications if AP_ROLE_ALLOWED is set to AP_INITIATOR while the XAP instance is in the AP_IDLE state. This situation occurs when the A_ASSOC_IND primitive has been queued prior to setting AP_ROLE_ALLOWED to AP_INITIATOR. To avoid this situation, the AP_ROLE_ALLOWED attribute should be set before the XAP instance is bound to a presentation address if it will only be used to initiate associations.

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.

AP_RSP_AEID

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

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_RSP_AEQ

The AP_RSP_AEQ is the responding AE-qualifier parameter of the AARE APDU.

See the description of the **ap_aeq_t** data type for information about how its absence is indicated.

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_RSP_APID

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

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_RSP_APT

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

This attribute is not used in the ‘‘X.410-1984’’ mode. Refer to the description of the AP_MODE_SEL attribute.

AP_SESS_AVAIL

The AP_SESS_AVAIL attribute indicates which versions of the Session Layer protocol are currently available.

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.

AP_SESS_OPT_AVAIL

The AP_SESS_OPT_AVAIL attribute is used to indicate which optional session capabilities are available through the XAP interface.

If the AP_UCBC flag is set, the underlying provider supports user control of basic concatenation. If the AP_UCEC flag is set, the underlying provider supports user control of extended concatenation. For further information, refer to *ap_snd*( ) and the Session Layer protocol specification.

AP_SFU_AVAIL

The AP_SFU_AVAIL attribute indicates which Session Layer functional units are currently available.

The value of this attribute may affect the value of AP_TOKENS_AVAIL.

AP_SFU_SEL

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

AP_STATE

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

AP_TOKENS_AVAIL

A token is an attribute of an association which is dynamically assigned to one user at a time. The user that possesses a token has exclusive rights to initiate the service which 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.

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 by AP_INIT_TOKENS.

The attribute descriptions above indicate when setting one attribute's value may affect the value of another attribute. These dependencies are summarised in the table below:

Attribute Name	Affects	Is Affected By
AP_BIND_PADDR	AP_LCL_PADDR	
AP_LCL_PADDR		AP_BIND_PADDR
AP_LIB_SEL	potentially all	
AP_SFU_SEL	AP_TOKENS_AVAIL	
AP_TOKENS_AVAIL		AP_SFU_SEL

The following table provides additional information about the XAP environment attributes. This information includes:

Attribute The symbolic constant defined in <**xap.h**> which is used to identify the attribute.

Type The data type of the values which are legal for the attribute.

Default The default value supplied for the attribute (if any).

Values If applicable, the set of values which are legal for the attribute. If no default value is supplied, the default value is given as ''none'' or ''not present''. ''None'' implies that a value must be specified by the user prior to issuance of a primitive, whereas ''not present'' implies that a value need not be specified, as the attribute represents an optional field of a particular primitive. They are otherwise identical.

Readable The states in which the attribute may be read using *ap_get_env*( ) (states are given as values of the AP_STATE attribute)

Writable The states during which the attribute may be assigned a value using either *ap_set_env*( ) or *ap_init_env*( ) (states are given as values of the AP_STATE attribute).

Attribute	Type/Values	Readable	Writable
AP_ACSE_AVAIL	unsigned long bit values: 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_AFU_AVAIL	unsigned long bit values: NULL	always	never
AP_AFU_SEL	unsigned long bit values: NULL default: NULL	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_BIND_PADDR	ap_paddr_t default: none	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_CLD_AEID	ap_aei_api_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLD_AEQ	ap_aeq_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLD_APID	ap_aei_api_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLD_APT	ap_apt_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLD_CONN_ID	ap_conn_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_CLG_AEID	ap_aei_api_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_AEQ	ap_aeq_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_APID	ap_aei_api_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE

Attribute	Type/Values	Readable	Writable
AP_CLG_APT	ap_apt_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_CLG_CONN_ID	ap_conn_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_COPYENV	long one of: TRUE FALSE default: FALSE	always	always
AP_CNTX_NAME	ap_objid_t default: none	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_DCS	ap_dcs_t	in any states but: AP_UNBOUND	never
AP_DIAGNOSTIC	ap_diag_t	always	never
AP_DPCN	ap_dcn_t default: not present	always	only in states: AP_UNBOUND AP_IDLE
AP_DPCR	long one of: AP_DPCR_NOVAL AP_ACCEPT AP_USER_REJ AP_PROV_REJ default: AP_DPCR_NOVAL	only in state: AP_WASSOCrsp_ASSOCind	only in state(s): AP_WASSOCrsp_ASSOCind
AP_FLAGS	unsigned long bit values: AP_NDELAY	always	always
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

Attribute	Type/Values	Readable	Writable
AP_INIT_TOKENS	unsigned long bit values (one per token): AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE AP_SYNCMINOR_TOK_REQ AP_SYNCMINOR_TOK_ACPT AP_SYNCMINOR_TOK_CHOICE AP_MAJACT_TOK_REQ AP_MAJACT_TOK_ACPT AP_MAJACT_TOK_CHOICE AP_RELEASE_TOK_REQ AP_RELEASE_TOK_ACPT AP_RELEASE_TOK_CHOICE default: AP_DATA_TOK_REQ | AP_SYNCMINOR_TOK_REQ | AP_MAJACT_TOK_REQ | AP_RELEASE_TOK_REQ	only in states: AP_UNBOUND AP_IDLE AP_WASSOCcnf_ASSOCreq AP_WASSOCrsp_ASSOCind	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	always	never
AP_LIB_SEL	unsigned long bit values: AP_LIBVER1 default: none	always	only in state: AP_UNBOUND
AP_MODE_AVAIL	unsigned long bit values: AP_NORMAL_MODE AP_X410_MODE	always	never
AP_MODE_SEL	unsigned long bit values: AP_NORMAL_MODE AP_X410_MODE default: AP_NORMAL_MODE	always	only in states: AP_UNBOUND AP_IDLE
AP_MSTATE	unsigned long bit values: AP_SNDMORE AP_RCVMORE	always	never
AP_OLD_CONN_ID	ap_old_conn_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE

Attribute	Type/Values	Readable	Writable
AP_OPT_AVAIL	unsigned long bit values: AP_NSAP_WILD AP_TSEL_WILD AP_SSEL_WILD AP_PSEL_WILD	always	never
AP_PCDL	ap_cdl_t default: not present	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind	only in states: AP_UNBOUND AP_IDLE
AP_PCDRL	ap_cdrl_t default: not present	only in states: AP_WASSOCrsp_ASSOCind AP_DATA_XFER	only in states: AP_WASSOCrsp_ASSOCind
AP_PFU_AVAIL	unsigned long bit values: NULL	always	never
AP_PFU_SEL	unsigned long bit values: NULL default: NULL	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_PRES_AVAIL	unsigned long bit values: 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_QLEN	long default: 0	always	only in state: AP_UNBOUND
AP_QOS	ap_qos_t	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

Attribute	Type/Values	Readable	Writable
AP_ROLE_CURRENT	unsigned long bit values: AP_INITIATOR AP_RESPONDER	in any states but: AP_UNBOUND AP_IDLE	never
AP_RSP_AEID	ap_aei_api_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_RSP_AEQ	ap_aeq_t default: not present	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_RSP_APID	ap_aei_api_id_t default: not present	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_RSP_APT	ap_apt_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	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
AP_SESS_OPT_AVAIL	unsigned long bit values: AP_UCBC AP_UCEC	always	never
AP_SFU_AVAIL	unsigned long or'd bits begin with LSB: AP_SESS_HALFDUPLEX AP_SESS_DUPLEX AP_SESS_XDATA AP_SESS_MINORSYNC AP_SESS_MAJORSYNC AP_SESS_RESYNC AP_SESS_ACTMGMT AP_SESS_NEGREL AP_SESS_CDATA AP_SESS_EXCEPT AP_SESS_TDATA AP_SESS_DATASEP	always	never

Attribute	Type/Values	Readable	Writable
AP_SFU_SEL	unsigned long or'd bits begin with LSB: AP_SESS_HALFDUPLEX AP_SESS_DUPLEX AP_SESS_XDATA AP_SESS_MINORSYNC AP_SESS_MAJORSYNC AP_SESS_RESYNC AP_SESS_ACTMGMT AP_SESS_NEGREL AP_SESS_CDATA AP_SESS_EXCEPT AP_SESS_TDATA AP_SESS_DATASEP default: AP_SESS_DUPLEX	always	only in states: AP_UNBOUND AP_IDLE AP_WASSOCrsp_ASSOCind
AP_STATE	unsigned long one of: AP_UNBOUND AP_IDLE AP_DATA_XFER AP_WASSOCrsp_ASSOCind AP_WASSOCcnf_ASSOCreq AP_WRELrsp_RELind AP_WRELcnf_RELreq AP_WRESYNrsp_RESYNind AP_WRESYNcnf_RESYNreq AP_WRELrsp_RELind_init AP_WRELcnf_RELreq_rsp AP_WACTDrsp_ACTDind AP_WACTDcnf_ACTDreq AP_WACTErsp_ACTEind AP_WACTEcnf_ACTEreq AP_WACTIrsp_ACTIind AP_WACTIcnf_ACTIreq AP_WSYNCMArsp_SYNCMAind AP_WSYNCMAcnf_SYNCMAreq AP_WCDATArsp_CDATAind AP_WCDATAcnf_CDATAreq AP_WRECOVERYind AP_WRECOVERYreq	always	never
AP_TOKENS_AVAIL	unsigned long bit values: AP_DATA_TOK AP_SYNCMINOR_TOK AP_MAJACT_TOK AP_RELEASE_TOK	in any states but: AP_UNBOUND AP_IDLE AP_WASSOCcnf_ASSOCreq AP_WASSOCrsp_ASSOCind	never
AP_TOKENS_OWNED	unsigned long bit values: AP_DATA_TOK AP_SYNCMINOR_TOK AP_MAJACT_TOK AP_RELEASE_TOK	in any states but: AP_UNBOUND AP_IDLE AP_WASSOCcnf_ASSOCreq AP_WASSOCrsp_ASSOCind	never

The following C types appear in the table above and are defined in <**xap.h**>:

ap_aeq_t is used to convey objects specified as ASN.1 type AE-qualifier and is defined as:

```
typedef struct {
    int size;              /* buffer size in bytes                    */
    unsigned char *udata;  /* buffer with user encoded AE-qualifier   */
} ap_aeq_t;
```

udata is a pointer to a buffer of user encoded AE-qualifier; *size* is the length of that buffer in octets.

ap_apt_t is used to convey objects specified as ASN.1 type AP-title and is defined as:

```
typedef struct {
    int size;              /* buffer size in bytes                    */
    unsigned char *udata;  /* buffer with user encoded AP-title       */
} ap_apt_t;
```

udata is a pointer to a buffer of user encoded AP-title; *size* is the length of that buffer in octets.

For optional PDU parameters of type, AE-qualifier or AP-title, setting *size* to –1 indicates that the parameter is not present. In addition, an optional parameter that corresponds to an environment attribute may be specified to be absent by invoking *ap_set_env*( ) with a NULL pointer as the *val* argument.

ap_aei_api_id_t is used to convey application entity/process identifier and is defined as:

```
typedef struct {
    int size;              /* buffer size in bytes                    */
    unsigned char *udata;  /* buffer with user encoded AE-identifier  */
                           /* or AP-identifier                        */
} ap_aei_api_id_t;
```

The **ap_aei_api_id_t** structure is used to convey application entity/process identifier values. Application entity/process identifier values are stored in their encoded form including *tag* and *length*. The absence of an application entity/process identifier parameter is indicated by setting the *size* field to –1.

ap_cdl_t is used to convey context definition lists and is defined as:

```
typedef struct {
    long  pci;                /* Presentation Context ID           */
    ap_objid_t   *a_sytx;     /* Abstract Syntax Object ID         */
    int  size_t_sytx;         /* Number of transfer syntaxes       */
    ap_objid_t  **m_t_sytx;   /* Array of ptrs to transfer syntaxes */
} ap_cdl_elt_t;

typedef struct {
    int   size;               /* Number of elements            */
    ap_cdl_elt_t  *m_ap_cdl;  /* Array                         */
} ap_cdl_t;
```

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

The presentation context identifier is represented by *m_ap_cdl[i].pci*. *m_ap_cdl[i].a_sytx* is a pointer to an **ap_objid_t** and is used to convey the abstract syntax name associated with the presentation context identifier. *m_ap_cdl[i].m_t_sytx* is a pointer to an array of type **ap_objid_t *** which includes the transfer syntaxes that the association-initiator is capable of supporting for the named abstract syntax. *m_ap_cdl[i].size_t_sytx* is the number of elements in the *m_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 to be absent by invoking *ap_set_env*( ) with AP_PCDL as the *attr* argument and a NULL pointer as the *val* argument.

ap_cdrl_t is used to convey context definition result lists and is defined as:

```
typedef struct {
    long  res;                  /* Result of negotiation        */
    ap_objid_t   *t_sytx;       /* Negotiated transfer syntax   */
    long  prov_rsn;             /* Reason for rejection         */
} ap_cdrl_elt_t;

typedef struct {
    int size ;                  /* Number of elements           */
    ap_cdrl_elt_t  *m_ap_cdl;   /* Array                        */
} ap_cdrl_t;
```

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

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 *m_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 AP_ACCEPT, AP_USER_REJ or AP_PROV_REJ. If *m_ap_cdrl[i].res* is set to AP_ACCEPT, then *m_ap_cdrl[i].t_sytx* indicates the transfer syntax selected by the association-responder. If *m_ap_cdrl[i].res* is set to either AP_USER_REJ or AP_PROV_REJ, *m_ap_cdrl[i].prov_rsn* is set to the reason for the rejection of the abstract syntax.

The possible values for *m_ap_cdrl[i].prov_rsn* are AP_RSN_NSPEC (reason not specified), AP_A_SYTX_NSUP (abstract syntax not supported), AP_PROP_T_SYTX_NSUP (proposed transfer syntaxes not supported), and AP_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 to be absent by invoking *ap_set_env*( ) with AP_PCDRL as the *attr* argument and a NULL pointer as the *val* argument.

ap_dcn_t is used to convey default context names and is defined as:

```
typedef struct {
     ap_objid_t  *a_sytx;  /* Abstract Syntax Name */
     ap_objid_t  *t_sytx;  /* Transfer Syntax Name */
} ap_dcn_t;
```

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 is used to convey the defined context set and is defined as:

```
typedef struct {
    long  pci;                  /* Presentation Context ID  */
    ap_objid_t   *a_sytx;       /* Abstract syntax          */
    ap_objid_t   *t_sytx;       /* Transfer syntax          */
} ap_dcs_elt_t;

typedef struct {
    int size ;                  /* Number of elements       */
    ap_dcs_elt_t  *dcs;         /* Array                    */
} ap_dcs_t;
```

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[i].pci*), an abstract syntax name (*dcs[i].a_sytx*), and a transfer syntax name (*dcs[i].t_sytx*).

ap_diag_t is used to convey additional diagnostic information which may be available from a particular implementation. It is defined as:

```
typedef struct {
    long rsn;                   /* reason for the abort    */
    long evt;                   /* event that caused abort */
    long src;                   /* source of abort         */
    char *error;                /* textual message         */
} ap_diag_t;
```

The *src* field will be set to indicate the source of an abort. Possible values for the *src* field are those defined for the *cdata->src* parameter to the A_PABORT_IND primitive, plus AP_ACSE_SERV_PROV (ACSE service provider).

The *rsn* field will be set to indicate the reason for the abort. The values for the *rsn* field depend on the value of the *src* field. If the *src* field is set to AP_ACSE_SERV_PROV then the *rsn* field will be set to one of the following:

AP_NSPEC	The reason for the abort is not specified.
AP_UNREC_APDU	Abort due to an unrecognised ACSE protocol data unit.
AP_UNEXPT_APDU	Abort due to an unexpected ACSE protocol data unit.
AP_UNREC_APDU_PARM	Abort due to an unrecognised ACSE protocol data unit parameter.
AP_UNEXPT_APDU_PARM	Abort due to an unexpected ACSE protocol data unit parameter.
AP_INVAL_APDU_PARM	Abort due to an invalid ACSE protocol data unit parameter.

If the *src* field is set to AP_PRES_SERV_PROV, the *rsn* field will be set to one of the corresponding reasons defined for the XAP primitive A_PABORT_IND.

If the *src* field is set to AP_SESS_SERV_PROV, the *rsn* field will be set to one of the corresponding reasons defined for the XAP primitive A_PABORT_IND.

If the *src* field is set to AP_TRAN_SERV_PROV, the *rsn* field will be set either to one of the corresponding reasons defined for the XAP primitive A_PABORT_IND, or to AP_TRAN_DISCONNECT (abort due to transport disconnect received).

The *evt* field is used to indicate the incoming event which caused the abort. If the *src* field is set to AP_ACSE_SERV_PROV, the APDU which triggered the abort will be indicated as follows:

AP_AEI_AARQ	Associate request APDU.
AP_AEI_AARE	Associate response APDU.
AP_AEI_RLRQ	Associate release request APDU.
AP_AEI_RLRE	Associate release response APDU.
AP_AEI_ABRT	Associate abort APDU.

If the *src* field is set to AP_PRES_SERV_PROV, the *evt* field will be set to one of the corresponding values defined for the XAP primitive A_PABORT_IND.

If the *src* field is set to AP_TRAN_SERV_PROV and the *rsn* field is set to AP_TRAN_DISCONNECT, the *evt* field will be set to an implementation-defined diagnostic value such as that returned by the XTI *t_rcvdis*( ) call.

The *error* field will be set up to point to a text string which describes the error condition, or will be set to NULL if no such text string is available under the currently defined *LOCALE*. The text string is in the natural language of the currently defined *LOCALE*.

ap_paddr_t is used to convey presentation addresses and is defined as:

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

typedef struct {
    ap_octet_string_t nsap;  /* NSAPAddress                         */
    int nsap_type;           /* AP_UNKNOWN,  AP_CLNS,  AP_CONS,     */
                             /* AP_RFC1006, other = system dependent */
} ap_nsap_t;

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_nsap_t *nsaps;               /* Array of network addresses   */
                                    /* and types                    */
} ap_paddr_t;
```

P_selector, *S_selector* and *T_selector* are pointers to octet strings corresponding to the presentation, session and transport selectors respectively. These octet strings are represented by the **ap_octet_string_t** structure. The *length* field of this structure indicates how many octets are in the octet string pointed by *data*. To specify a null selector value, the length field of the **ap_octet_string_t** structure is set to 0. For *selector wildcard* presentation addresses, non-specified selector values are indicated by setting the corresponding pointer(s) in the **ap_paddr_t** structure to NULL.

The *n_nsaps* element is used to specify how many network address components are in the array, *nsaps*. Each element of the *nsaps* array is an **ap_octet_string_t** structure with an associated *nsap_type* which identifies the type of network (or system dependent values) to which the NSAP refers. An *nsap_type* of AP_UNKNOWN indicates that the network type is not known.

When multiple network address components are included in a presentation address, the specific network address(es) chosen by the provider and the manner by which it is selected for initiating

or listening to connections is not specified by XAP and is a local implementation issue. When used to represent a wildcard address, the value of *n_nsaps* may be zero. In this case, all locally defined network addresses are implied. In all other cases, *n_nsaps* must be a positive value.

Presentation addresses are restricted as follows:

- Session selectors cannot exceed 16 octets.
- Transport selectors cannot exceed 32 octets.
- Remote addresses may not have more than 8 Network addresses.
- Individual Network addresses cannot exceed 20 octets.

Note that implementations may impose further restrictions on local addresses.

ap_conn_id_t is used to convey session connection identifiers and is defined as:

```
typedef struct {
    ap_octet_string_t  *user_ref;      /* SS-user Ref.              */
    ap_octet_string_t  *comm_ref;      /* Common Ref.               */
    ap_octet_string_t  *addtl_ref;     /* Additional Ref.           */
} ap_conn_id_t;
```

Each of the members of the **ap_conn_id_t** structure are of type **ap_octet_string_t** (see above). 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**.

```
typedef struct {
    ap_octet_string_t  *clg_user_ref;  /* calling SS-user reference */
    ap_octet_string_t  *cld_user_ref;  /* called SS-user reference  */
    ap_octet_string_t  *comm_ref;      /* common reference          */
    ap_octet_string_t  *addtl_ref;     /* Additional Reference      */
} ap_old_conn_id_t  ;
```

Each of the members of the **ap_old_conn_id** structure is of type **ap_octet_string_t**. The *clg_user_ref*, *cld_user_ref* and *comm_ref* members must be <= 64 octets in length. The *addtl_ref* member must be <= 4 octets. The absence of a particular member of an old connection identifier may be indicated either by setting the corresponding field to NULL, or by specifying a 0 length **ap_octet_string_t**.

ap_objid_t is used to convey objects of ASN.1 type OBJECT IDENTIFIER and is defined as follows:

```
#define AP_MAXOBJBUF  12
#define AP_CK_OBJ_NULL(O)          ((O)->length ? 0 : 1)
#define AP_SET_OBJ_NULL(O)         ((O)->length = 0)

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

The **ap_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.

Routines to create and operate on objects of this type may be provided as part of an Encode / Decode Library but such routines are not part of the XAP specification.

The **ap_qos_t** structure is used to convey quality of service requirements and is defined as follows:

```
#define AP_NO   0
#define AP_YES  1

typedef struct {
    ap_thrpt_t  throughput;     /* throughput                     */
    ap_transdel_t transdel;     /* transit delay                  */
    ap_rate_t   reserrorrate;   /* residual error rate            */
    ap_rate_t   transffailprob; /* transfer failure probability   */
    ap_rate_t   estfailprob;    /* connection establ failure      */
                                /* probability                    */
    ap_rate_t   relfailprob;    /* connection release failure     */
                                /* probability                    */
    ap_rate_t   estdelay;       /* connection establishment delay */
    ap_rate_t   reldelay;       /* connection release delay       */
    ap_rate_t   connresil;      /* connection resilience          */
    unsigned int protection;    /* protection                     */
    int         priority;       /* priority: AP_PRITOP,           */
                                /* AP_PRIHIGH, AP_PRIMID,         */
                                /* AP_PRILOW, or AP_PRIDFLT       */
    char        optimizedtrans; /* optimized dialogue transfer    */
                                /* value: AP_YES or AP_NO         */
    char        extcntl;        /* extended control: AP_YES or    */
                                /* AP_NO                          */
} ap_qos_t;
```

The fields of the **ap_qos_t** structure specify the ranges of acceptable values for the quality of service requirements of an association. The supporting structures are as follows:

```
typedef struct {
    ap_reqvalue_t   maxthrpt;       /* maximum throughput */
    ap_reqvalue_t   avgthrpt;       /* average throughput */
} ap_thrpt_t;

typedef struct {
    ap_reqvalue_t   maxdel;         /* maximum transit delay */
    ap_reqvalue_t   avgdel;         /* average transit delay */
} ap_transdel_t;

typedef struct {
    long            targetvalue;    /* target value             */
    long            minacceptvalue; /* limiting acceptable value */
} ap_rate_t;

typedef struct {
    ap_rate_t       called;         /* called rate  */
    ap_rate_t       calling;        /* calling rate */
} ap_reqvalue_t;
```

Values are assigned to attributes either through *ap_init_env*( ), *ap_restore*( ), *ap_set_env*( ) or during *ap_rcv*( ) and *ap_snd*( ) event processing. The environment attributes are initialized to their

default values when an *ap_init_env*( ) call is made. The attributes can be changed by the user using the *ap_set_env*( ) call. Also, the attributes can be changed by the library when needed due to *ap_rcv*( ) and *ap_snd*( ) calls. The *ap_restore*( ) call will only set the attributes to the values that they were when they were saved using the *ap_save*( ) call. When *ap_get_env*( ) is issued for an attribute that has no value assigned, the location pointed at by *aperrno_p* will be set to the [AP_BADENV] error code.

Chapter 4

XAP Functions

This chapter presents the *manual pages* for the XAP API. These define the functions which make up XAP, providing the detailed specifications of parameters and data structures.

The *manual pages* for *ap_snd*( ) and *ap_rcv*( ) include tables which define the valid states in which each primitive can be sent or received, the resulting state, and the effect on the variables that control the protocol's operation.

4.1 Introduction

4.1.1 Functions

This section describes the functions in XAP. A complete list of these functions is provided below.

ap_bind( )	Associate a Presentation Address with an instance of XAP.
ap_close( )	Close an instance of XAP.
ap_error( )	Return an error message.
ap_free( )	Free memory for XAP data structures.
ap_get_env( )	Get the value of an XAP environment attribute.
ap_init_env( )	Initialise an instance of XAP.
ap_ioctl( )	Control the generation of software interrupts.
ap_look( )	Examine the next ACSE / Presentation primitive from the association / connection.
ap_open( )	Create an instance of XAP.
ap_poll( )	Input / output multiplexing.
ap_save( )	Save an instance of XAP.
ap_set_env( )	Set the value of an XAP environment attribute.
ap_snd( )	Send an ACSE / Presentation primitive over the association / connection.
ap_rcv( )	Receive an ACSE / Presentation primitive from the association / connection.
ap_restore( )	Restore an instance of XAP environment.

4.1.2 Errors

Most of these functions have one or more possible error returns. The **Return Value** section of each XAP manual page indicates how the occurrence of an error is signalled to the user. For most functions, an error condition is indicated by a returned value of −1, and the location pointed at by *aperrno_p* parameter is set to the error code indicating the error condition.

Each function description includes a list of error conditions that are reported by XAP. In addition, errors reported by underlying service providers (i.e., ACSE, Presentation, Session, Transport, ASN.1, or the operating system) may be passed through to the user. The *class* of a particular error can be determined by examining the two least significant octets of the error code after shifting right 16 bits. The following numbers identify the defined error classes:

```
/*
 * These ID numbers for each protocol are used to distinguish
 * #defines of various kinds for each layer, such as primitive
 * names, environment attribute names, error codes, etc.
 */

#define AP_ASN1_ID (11)
#define AP_ID       (8)
#define AP_ACSE_ID (7)
#define AP_PRES_ID (6)
#define AP_SESS_ID (5)
```

```
#define AP_TRAN_ID (4)
#define AP_OS_ID   (0)
```

Below is a complete list of errors that are reported by XAP. Error codes associated with errors reported from underlying service providers are dependent on the particular provider in question. Refer to the interface specifications for those providers for further information.

A/P-LIBRARY ERRORS

[AP_ACCES]	Request to bind to specified address denied.
[AP_AGAIN]	Request not completed.
[AP_AGAIN_DATA_PENDING]	XAP was unable to complete the requested action. Try again. There is an event available for the user to receive.
[AP_BADATTRVAL]	Bad value for environment attribute.
[AP_BADALLOC]	The ap_user_alloc/ap_user_dealloc argument combination was invalid.
[AP_BADASLSYN]	The transfer syntaxes proposed for the ACSE syntax are not supported.
[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_RES]	Cdata field value invalid: res.
[AP_BADCD_RESYNC_TYPE]	Cdata field value invalid: resync_type.
[AP_BADCD_RSN]	Cdata field value invalid: rsn.
[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_BADDATA]	User data not allowed on this service.
[AP_BADENV]	A mandatory attribute is not set.
[AP_BADF]	Not a presentation service endpoint.
[AP_BADFLAGS]	The specified 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.
[AP_BADNSAP]	The format of the NSAP portion of the Presentation Address is not supported.
[AP_BADPARSE]	Attribute parse failed.

[AP_BADPRIM]	Unrecognised primitive from user.
[AP_BADRESTR]	Attributes not restored due to more bit on.
[AP_BADROLE]	Request invalid due to value of AP_ROLE.
[AP_BADSAVE]	Attributes not saved due to more bit on.
[AP_BADSAVEF]	Invalid FILE pointer.
[AP_BADUBUF]	Bad length for user data.
[AP_DATA_OVERFLOW]	User data and presentation service pci exceeds 512 bytes on session V1 or the length of user data exceeds a locally defined limit, as stated in the CSQ.
[AP_HANGUP]	Association closed or aborted.
[AP_LOOK]	A pending event requires attention.
[AP_NOATTR]	No such attribute.
[AP_NOBUF]	Could not allocate enough buffers.
[AP_NODATA]	An attempt was made to send a primitive with no user data.
[AP_NOENV]	No environment for that fd.
[AP_NOMEM]	Could not allocate enough memory.
[AP_NOREAD]	Attribute is not readable.
[AP_NOWRITE]	Attribute is not writable.
[AP_NO_PRECEDENCE]	The resynchronisation requested by the local user does not have precedence over the one requested by the remote user.
[AP_NOT_SUPPORTED]	The action requested is not supported by this implementation of XAP.
[AP_PDUREJ]	Invalid PDU rejected.
[AP_SUCCESS_DATA_PENDING]	The requested action was completed successfully. There is an event available for the user to receive.

4.1.3 Structure Definitions

The following are definitions for the **ap_cdata_t** and **ap_a_assoc_env_t** structures. These definitions shown here are referenced in subsequent manual pages.

```
typedef struct {
  long udata_length;             /* length of user-data field              */
  long rsn;                      /* reason for activity/abort/release prim */
  long evt;                      /* event that caused abort                */
  long sync_p_sn;                /* synchronization point serial number    */
  long sync_type;                /* synchronization type                   */
  long resync_type;              /* resynchronization type                 */
  long src;                      /* source of abort                        */
  long res;                      /* result of association/release request  */
  long res_src;                  /* source of result                       */
  long diag;                     /* reason for association rejection       */
  unsigned long tokens;          /* tokens identifier: 0=>"tokens absent"  */
  ap_a_assoc_env_t *env;         /* environment attribute values           */
  ap_octet_string_t act_id;      /* activity identifier                    */
  ap_octet_string_t old_act_id;  /* old activity identifier                */
  ap_old_conn_id_t *old_conn_id; /* old session connection identifier      */
} ap_cdata_t;
```

```
typedef struct {
    unsigned long mask;             /* bit mask        */
    unsigned long mode_sel;         /* AP_MODE_SEL     */
    ap_objid_t cntx_name;           /* AP_CNTX_NAME    */
    ap_aei_api_id_t clg_aeid;       /* AP_CLG_AEID     */
    ap_aeq_t clg_aeq;               /* AP_CLG_AEQ      */
    ap_aei_api_id_t clg_apid;       /* AP_CLG_APID     */
    ap_apt_t clg_apt;               /* AP_CLG_APT      */
    ap_aei_api_id_t cld_aeid;       /* AP_CLD_AEID     */
    ap_aeq_t cld_aeq;               /* AP_CLD_AEQ      */
    ap_aei_api_id_t cld_apid;       /* AP_CLD_APID     */
    ap_apt_t cld_apt;               /* AP_CLD_APT      */
    ap_paddr_t rem_paddr;           /* AP_REM_PADDR    */
    ap_cdl_t pcdl;                  /* AP_PCDL         */
    ap_dcn_t dpcn;                  /* AP_DPCN         */
    ap_qos_t qos;                   /* AP_QOS          */
    unsigned long a_version_sel;    /* AP_ACSE_SEL     */
    unsigned long p_version_sel;    /* AP_PRES_SEL     */
    unsigned long s_version_sel;    /* AP_SESS_SEL     */
    unsigned long afu_sel;          /* AP_AFU_SEL      */
    unsigned long pfu_sel;          /* AP_PFU_SEL      */
    unsigned long sfu_sel;          /* AP_SFU_SEL      */
    ap_conn_id_t *clg_conn_id;      /* AP_CLG_CONN_ID  */
    ap_conn_id_t *cld_conn_id;      /* AP_CLD_CONN_ID  */
    unsigned long init_sync_pt;     /* AP_INIT_SYNC_PT */
    unsigned long init_tokens;      /* AP_INIT_TOKENS  */
    ap_aei_api_id_t rsp_aeid;       /* AP_RSP_AEID     */
    ap_aeq_t rsp_aeq;               /* AP_RSP_AEQ      */
    ap_aei_api_id_t rsp_apid;       /* AP_RSP_APID     */
    ap_apt_t rsp_apt;               /* AP_RSP_APT      */
    ap_cdrl_t pcdrl;                /* AP_PCDRL        */
    long  dpcr;                     /* AP_DPCR         */
} ap_a_assoc_env_t;
```

4.1.4 Token Assignment

This section defines the values which are used to specify the assignment of tokens during association establishment and during resynchronisation. Values are formed by OR'ing together bits corresponding to the requested position for each token.

The requesting user may set the following values:

Token	Allowed Values
Data	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT AP_DATA_TOK_CHOICE
Synchronize-minor	one of: AP_SYNCMINOR_TOK_REQ AP_SYNCMINOR_TOK_ACPT AP_SYNCMINOR_TOK_CHOICE
Major/Activity	one of: AP_MAJACT_TOK_REQ AP_MAJACT_TOK_ACPT AP_MAJACT_TOK_CHOICE
Release	one of: AP_RELEASE_TOK_REQ AP_RELEASE_TOK_ACPT AP_RELEASE_TOK_CHOICE

Assignments for tokens that are unavailable are ignored. If no assignment is given for an available token, AP_XXXXX_TOK_REQ is specified by the service.

The accepting user may set the following values:

Token	Allowed Values
Data	one of: AP_DATA_TOK_REQ AP_DATA_TOK_ACPT
Synchronize-minor	one of: AP_SYNCMINOR_TOK_REQ AP_SYNCMINOR_TOK_ACPT
Major/Activity	one of: AP_MAJACT_TOK_REQ AP_MAJACT_TOK_ACPT
Release	one of: AP_RELEASE_TOK_REQ AP_RELEASE_TOK_ACPT

Assignments for tokens that are unavailable or that were assigned to one side or the other by the requestor are ignored. If no assignment is given for an *acceptor's choice* token, AP_XXXXX_TOK_REQ is specified by the service.

NAME

ap_bind - bind a Presentation Address to an XAP instance

SYNOPSIS

```
#include <xap.h>

int ap_bind (
    int fd,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function associates the Presentation Address stored in the AP_BIND_PADDR environment variable with the XAP Instance specified by *fd*. Upon successful completion the service provider may begin enqueuing incoming associations or sending outbound association requests. All necessary environment variables (e.g., AP_BIND_PADDR, AP_ROLE_ALLOWED) should be set prior to calling *ap_bind*( ).

When this function is called, an attempt may be made to bind to the specified address. As a part of the bind procedure, an authorisation check may be performed to verify that all of the processes that share this XAP instance are authorised to use the new address. If all are authorised to do so, the bind request will succeed and the XAP instance may be used to send (receive) primitives from (addressed to) the new address. Successfully calling this function causes the state of XAP to move from AP_UNBOUND to AP_IDLE.

Some implementations may perform no authorisation checking. In this case the [AP_ACCES] error response will not be generated. Other implementations may defer binding and authorisation until an A_ASSOC_REQ or A_ASSOC_RSP primitive is issued. In this case, if the authorisation check fails, the AP_ACCES error will be returned by *ap_snd( )*. The local address can then be changed to an acceptable value and the primitive reissued, or the connection can be closed.

An instance can be bound to a presentation address only if all of the processes that share it are authorised to use the requested address. Consequently, when an attempt is made to bind an address, the effective UIDs of all of the processes that share this instance of XAP may be checked against the list of users allowed to use the requested address. If all are authorised to use the address, *ap_bind( )* succeeds and the instance is bound to the specified presentation address. On the other hand, if any of the processes is not authorised to use the requested address, *ap_bind( )* fails and the instance remains unbound.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed to by *aperrno_p* is set to indicate the error.

ERRORS

[AP_ACCES]	Request to bind to specified address denied.
[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_BADNSAP]	The format of the NSAP portion of the Presentation Address is not supported.
[AP_NOENV]	There is no XAP environment associated with *fd*.

NAME

ap_close - close an XAP instance

SYNOPSIS

```
#include <xap.h>

int
ap_close (
    int fd,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function frees the resources allocated to support the instance of XAP identified by *fd*.

If the last *close* of the XAP instance occurs while an association is still active, the association (and any primitive that is being sent or received in multiple parts using the AP_MORE bit) is aborted before the resources are released.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed to by *aperrno_p* is set to indicate the error.

ERRORS

[AP_BADF] The *fd* parameter does not identify an XAP instance.

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

NAME

ap_error - return an error message

SYNOPSIS

```
#include <xap.h>

char *ap_error (
    unsigned long aperrno)
```

DESCRIPTION

This function returns a pointer to a message that describes the error indicated by *aperrno*. 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 the XAP Functions introduction (see Section 4.1 on page 56). The message pointer points to an internal buffer area. If the caller wishes to retain the message text, before calling *ap_error*( ) again, then the text should be copied to some private storage.

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

RETURN VALUE

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

ERRORS

No error conditions are reported by this function.

NAME

ap_free - free user memory associated with XAP data structures

SYNOPSIS

```
#include <xap.h>

int ap_free (
    int fd,
    unsigned long kind,
    void *val,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function frees memory for values of XAP environment attributes and data structures allocated on the XAP instance identified by fd. Common uses of this function would be to free memory allocated for the value of an environment attribute by XAP following an ap_get_env( ) invocation, and to free an ap_a_assoc_env_t structure returned with an A_ASSOC_IND by an ap_rcv invocation when the AP_COPYENV environment attribute is set. The argument *kind* identifies the kind of structure that is to be freed. Legal values for this argument are:

- #define identifiers associated with environment attributes whose type is not a long. These are the names listed in the "Attribute" column of the environment attribute summary table given in Chapter 3 (for example, AP_INIT_TOKENS).
- #define identifiers associated with data structures that are used to represent the values of environment attributes. These data structures are those listed in the Type/Value column of the environment attribute summary table given in Chapter 3. The identifier for a data structure is derived from the typedef name by converting it to upper case.
- #define identifiers associated with certain important C data structures not associated with environment attributes. These are as follows:

C Data Structure	Identifier
ap_cdata_t	AP_CDATA_T
ap_a_assoc_env_t	AP_A_ASSOC_ENV_T
ap_osi_vbuf_t	AP_OSI_VBUF_T

- The #define identifier AP_BUFFERS for a chain of user data buffers.

The argument *val* must be a pointer to a structure of the type indicated by *kind*.

The behaviour of *ap_free*( ) differs when releasing buffers and when releasing memory.

The user can release buffers by calling *ap_free*( ) with *kind* set to AP_BUFFERS or AP_OSI_VBUF_T. An *ap_free*( ) call with *kind* set to AP_BUFFERS releases ALL buffers in the supplied chain (including the first osi_vbuf/osi_dbuf/data buffer in the chain) by passing the entire chain to the user dealloc function. An *ap_free*( ) call with *kind* set to AP_OSI_VBUF_T releases all buffers in the chain EXCEPT the first osi_vbuf/osi_dbuf/data buffer, and sets *bcont* in the first osi_vbuf to NULL. Note that because all freeing of buffers is performed by the user dealloc function, osi_dbufs and buffers cannot be freed in isolation.

When releasing memory, the *ap_free*( ) function follows and frees all internal pointers. The top level structure (the structure pointed to by *val*) is not freed. If the user has supplied a deallocation function on the *ap_open*( ) call then:

- the *ap_user_dealloc*( ) function is called to free any buffers, and buffer chains, passed to *ap_free*( )

- if the AP_BUFFERS_ONLY flag was not set on the *ap_open* call, then the *ap_user_dealloc*( ) function will be called to free the memory comprising data structures pointed at by members of the data structure passed to *ap_free*( ).

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_BADFREE]	The function could not free the structure.
[AP_BADKIND]	The *kind* argument does not identify a known structure type.

NAME

ap_get_env - get the value of an XAP environment attribute

SYNOPSIS

```
#include <xap.h>

int ap_get_env (
    int fd,
    unsigned long attr,
    void *val,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function retrieves the value of an environment attribute for an XAP instance identified by *fd*. *Attr* is used to pass the ''name'' of the attribute to be retrieved, defined by the associated **#define** in the **<xap.h>** header file.

The value supplied as the *val* argument to this function depends upon 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 (e.g., **ap_dcs_t**), XAP allocates additional memory by calling the user supplied or built-in allocation routine, and assigns proper values to the pointer elements as required. Memory allocated by *ap_get_env( )* can be freed with the *ap_free( )* function.

Refer to *ap_env*( ) for information about attribute types.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

[AP_BADENV]	There is no value assigned to the requested environment attribute.
[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_NOATTR]	The *attr* argument does not specify a valid attribute type.
[AP_NOENV]	There is no XAP environment associated with *fd*.
[AP_NOMEM]	XAP could not allocate sufficient memory to copy the value of the specified attribute.
[AP_NOREAD]	The specified attribute is not readable.

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

NAME

ap_init_env - establish an instance of XAP and initialise the XAP environment

SYNOPSIS

```
#include <xap.h>

int ap_init_env (
    int fd,
    const char *env_file,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function initialises an otherwise uninitialised XAP instance identified by *fd*. In addition, *ap_init_env*( ) may be used to set the values of several writable environment attributes with a single function call rather than using *ap_set_env*( ) to set each attribute individually.

If no environment exists when *ap_init_env*( ) is called, memory will be allocated for the environment attributes in the calling process's data space and the attributes will be set to their default values (see the environment description in Chapter 3). If the user wishes to override the defaults for certain writable attributes, values for those attributes may be specified in an initialisation file as described below.

If an environment already exists when *ap_init_env*( ) is called, attributes will be assigned values. In this case, attributes will not automatically be set to their default values.

To set the environment attributes from values stored in a file, env_file must point to a null-terminated string that is the initialisation file's pathname. An environment initialisation file is generated by processing an **ap_env_file** file using the *ap_osic* command. Setting env_file to NULL indicates that no values are to be taken from an environment initialisation file. No error is reported if the file identified by env_file is zero length.

The environment initialisation file may contain assignments for any or all of the attributes that are writable in the current state. The *ap_init_env*( ) call will fail if env_file contains an assignment for an attribute that is not writable in the current state. Attributes that are not included in the file will not be modified. The following specific points about initialisation of the environment from a file should be noted:

- The values of read-only attributes are not affected by *ap_init_env*( ). The one exception to this is AP_LCL_PADDR. Since AP_LCL_PADDR is set as a side effect of setting AP_BIND_PADDR, its value may change as an indirect result of invoking *ap_init_env*( ) if the value of AP_BIND_PADDR is modified.
- If the environment file assigns a value to the AP_BIND_PADDR attribute, it should be noted that the Presentation address set will not be used until the *ap_bind*( ) function is called. Calling *ap_bind*( ) after *ap_init_env*( ) will cause the Presentation address to be validated and if the authorisation check succeeds, then the endpoint will be moved to the AP_IDLE state. If the *ap_init_env*( ) function is called after a successful *ap_bind*( ), then if a new Presentation address is assigned to AP_BIND_PADDR, the endpoint will not be re-bound until *ap_bind*( ) is called again.

The *flags* argument to this function is currently unused.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

CAVEAT

The output from the *ap_osic* command of one XAP implementation is not necessarily readable by the *ap_init_env*( ) function of another XAP implementation, as the format of the intermediate file is not defined. Environment initialisation files are therefore only guaranteed to be portable in the **ap_env_file** form.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, if the call fails because of an attempt to either set an attribute to an illegal value or set an attribute in an illegal state, *ap_init_env* will return the value of the symbolic constant that identifies that attribute. If the call fails for some other reason, a value of –1 is returned. Either type of failure will result in the location pointed at by *aperrno_p* being set to indicate the error.

ERRORS

[AP_BADATTRVAL]	An invalid value assignment was found in the initialisation file.
[AP_BADF]	The *fd* argument does not identify an XAP instance.
[AP_NOATTR]	An invalid attribute type appears in the initialisation file.
[AP_NOMEM]	XAP could not allocate sufficient memory to create an environment.
[AP_NOT_SUPPORTED]	The use of a non-NULL value for *env_file* is not supported by this implementation of XAP.
[AP_NOWRITE]	An assignment for an attribute that is not writable in the current state appears in the initialisation file.

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

NAME

ap_ioctl - control the generation of software interrupts

SYNOPSIS

```
#include <xap.h>

int ap_ioctl (
    int fd,
    int request,
    ap_val_t argument,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function controls the generation of software interrupts for the XAP instance identified by *fd*.

Software interrupt is an asynchronous mechanism that can be used to inform an application of pending events or state changes for a service that it is using. An XAP implementation may use such a mechanism to inform the XAP user about incoming events such as a primitive available to be received, or outgoing events such as a flow control restriction being lifted. This mechanism is complementary to the *ap_poll*( ) function which provides synchronous notification of such events.

Support for software interrupt and the mechanism by which such an interrupt is signalled to the XAP user is a feature of the operating system platform on which the XAP implementation runs. XAP does not define a software interrupt interface mechanism itself. Further, support for the function provided by *ap_ioctl*( ) is not a mandatory part of XAP and if not available, *ap_ioctl*( ) returns [AP_NOT_SUPPORTED]. Therefore a portable application should not rely on provision of this mechanism.

If the user requires software interrupts to be generated when a data event occurs, then *ap_ioctl*( ) should be called with a *request* parameter of AP_SETPOLL. The *l* member of the *ap_val_t argument* parameter is a bitmask used to indicate which events should generate a software interrupt. All XAP Library implementations recognise the following events:

AP_POLLRDNORM	Data (for example, an XAP Library primitive or user data associated with an XAP Library primitive) has arrived on the normal data flow and is available to be read.
AP_POLLRDBAND	Data has arrived outside the normal data flow and is available to be read. In implementations that do not support multiple data bands, this event will result in the same action as the AP_POLLRDNORM event.
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.
AP_POLLWRNORM	The same as AP_POLLOUT.
AP_POLLWRBAND	Out-of-band data can be sent. This event will be treated as AP_POLLOUT in implementations that do not support multiple data bands.

Support for events other than those listed above is optional. Users interested in developing applications that are portable across different XAP Library implementations should keep this caveat in mind.

If the user requires software interrupts to be disabled then *ap_ioctl*( ) should be called with a *request* parameter of AP_SETPOLL and the *l* member of the *ap_val_t* argument value as 0.

If the user wishes to obtain the current settings of the software interrupts bitmask then *ap_ioctl*( ) should be called with a request parameter of AP_GETPOLL. The *v* member of the *ap_val_t* argument must point to a location where the current setting of the bitmask is to be written by *ap_ioctl*( ).

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value 0 is returned. Otherwise, a value of -1 is returned and the location pointed to by *aperrno_p* is set to indicate the error.

ERRORS

[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_NOT_SUPPORTED]	*ap_ioctl*( ) operation is not supported by this implementation of XAP.

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

NAME

ap_look - examine next ACSE / Presentation primitive from the association / connection

SYNOPSIS

```
#include <xap.h>

int ap_look (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function is used to examine an indication or confirmation primitive without affecting the state of XAP.

fd identifies the XAP instance for which the user wishes to examine primitives.

When *ap_look* is called, *sptype* must point to an **unsigned long**, and *cdata* must point to an **ap_cdata_t** structure.

Upon return, the value of the **unsigned long** pointed to by *sptype* will indicate the type of primitive that is currently ready to be received.

Protocol information associated with a primitive will be conveyed by the **ap_cdata_t** structure pointed to by *cdata*. The value returned in *sptype* serves as the discriminant for what members of the *cdata* are affected. A complete discussion of the use of the *cdata* parameter is provided for each XAP primitive in Chapter 7.

A successful call to *ap_look*( ) always returns all the protocol information of a primitive (i.e. those fields in *cdata*) and may return some or all of the user data associated with that primitive.

Repeated calls will return the same protocol information, the same user data and may return further user data if the primitive was incomplete (AP_MORE flag was returned by the previous call) until such time as some or all of the pending primitive is removed. It is a local matter whether an implementation returns further user data on the subsequent call.

User-data received with a primitive will be returned to the user in the *ubuf* parameter. The XAP interface supports a vectored buffering scheme for handling user data. All data buffers are passed to XAP by the user in a chain of one or more *ap_osi_vbuf_t/ap_osi_dbuf_t* pairs. If there are not sufficient buffers, and *ap_look*( ) was called with the AP_ALLOC flag set, XAP will use the user-supplied buffer allocation routines. If however, the XAP-user failed to supply these routines in the *ap_open*( ) call, *ap_look*( ) returns –1, with the location pointed at by *aperrno_p* set to [AP_BADFLAGS]. If the AP_ALLOC flag is not set, and the (user) data buffer(s) are filled before completion of processing by XAP, *ap_look*( ) returns with the AP_MORE flag set. *ubuf* must point to a location holding a pointer to an *ap_osi_vbuf_t* structure, defined as follows:

```
typedef struct {
     unsigned char *db_base;     /* beginning of buffer    */
     unsigned char *db_lim;      /* last octet+1 of buffer */
     unsigned char db_ref;       /* reference count        */
} ap_osi_dbuf_t ;

typedef struct ap_osi_vbuf ap_osi_vbuf_t;
struct ap_osi_vbuf {
     ap_osi_vbuf_t *b_cont;          /* next message block */
     unsigned char *b_rptr;          /* 1st octet of data  */
     unsigned char *b_wptr;          /* 1st free location  */
     ap_osi_dbuf_t *b_datap;         /* data block         */
} ;
```

User-data associated with XAP primitives is returned in a linked list of message blocks. Each message block is represented by an *ap_osi_vbuf_t* structure and is associated with a data block. Data blocks, which are represented by *ap_osi_dbuf-t* structures, may be associated with more than one message block. The *db_ref* field of the *ap_osi_dbuf_t* structure indicates the number of *ap_osi_vbuf_t* structures that reference a particular data block. The *db_base* and *db_lim* fields of the *ap_osi_dbuf_t* structure point to the beginning and end of the data block respectively. The *b_rptr* and *b_wptr* fields of the referencing *ap_osi_vbuf_t* structures point to the first octets to be read and written within the data block respectively. The *b_cont* field of the *ap_osi_vbuf_t* points to the next message block in the chain or is NULL if this is the end of the list.

The user allocation routine is responsible for setting up all fields of the *ap_osi_vbuf_t* and *ap_osi_dbuf_t* structures when allocating buffers. If buffers are allocated by another mechanism, the user must ensure that the fields of each *ap_osi_vbuf_t* and *ap_osi_dbuf_t* pair in the chain are set up prior to calling *ap_look*( ).

ap_look( ) places data into any buffer where write space is available (b_wptr < db_lim) and updates b_wptr - no other fields in the *ap_osi_vbuf_t/ap_osi_dbuf-t* structures are updated (with the exception of *b_cont*, which is updated when adding further *ap_osi_vbuf_t/ap_osi_dbuf_t* pairs to the chain).

The user may pass full, partially full and empty receive buffers to *ap_look*( ). The user is responsible for ensuring that it is valid for the XAP library to fill any of the supplied buffers from b_wptr to db_lim.

If the user wishes all the buffers for *ap_look*( ) to be allocated using the user allocation routine, then the ubuf pointer will point to a NULL *ap_osi_vbuf_t* pointer.

The XAP user is responsible for decoding the user data received in the *ubuf* parameter. The general rules for decoding user data are stated here, please see individual manual pages in Chapter 4 for specific exceptions to these rules.

- If the ‘‘X.410-1984’’ mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the contents of *ubuf* buffer(s) are mapped directly from the SS-user data parameter of the equivalent session service primitive. Refer to the ISO Presentation Layer Protocol Specification (see reference **ISO 8823**) for further information concerning the encoding of these values. (The primary exception to this rule is the A-ASSOC primitives).
- If the ‘‘X.410-1984’’ mode of operation is *not* in effect and the primitive received is an ACSE primitive, the data received in the *ubuf* buffer(s) must be decoded according to the definition specified in the ACSE Protocol Specification (reference **ISO 8650**):

  ```
  [30] IMPLICIT SEQUENCE OF EXTERNAL
  ```

- If the ‘‘X.410-1984’’ mode of operation is *not* in effect and the primitive received is a Presentation primitive, the data received in the *ubuf* buffer(s) must be decoded according to the User-data definition specified in the Presentation Protocol Specification (reference **ISO 8823**):

```
CHOICE {
    [APPLICATION 0] IMPLICIT OCTET STRING,
    [APPLICATION 1] IMPLICIT SEQUENCE OF PDV-list
}
```

The *flags* argument is a bit mask used to control certain aspects of XAP processing. Values for this field are formed by OR’ing together zero or more of the following flags:

Flag	Description
AP_ALLOC	If AP_ALLOC is set and the user did not specify an allocation routine on *ap_open*( ) (or *ap_restore*( )) then –1 is returned and the location pointed to by *aperrno_p* is set to the [AP_BADFLAGS] error code. If no space is available in the supplied buffer chain (or the location pointed to by *ubuf* contains NULL) and either AP_ALLOC is not set or AP_ALLOC is set but the user allocation routine refuses to supply any buffers, then the call to *ap_look*( ) fails, –1 is returned and the location pointed to by *aperrno_p* is set to the [AP_NOBUF] error code. Regardless of the setting of AP_ALLOC, the user must have set the location pointed to by ubuf either to NULL or to point to a chain of one or more buffers. The AP_ALLOC flag setting only takes effect when any supplied buffers have been filled and more data remains to be returned to the user: • If the AP_ALLOC flag is set and space was available in the supplied buffer chain, the user allocation routine is called to supply further buffers as they are needed. If the user allocation routine refuses to supply further buffers then the AP_MORE flag is set and the call to *ap_look*( ) completes; 0 is returned. The user must free any buffers allocated by the user allocation routine either by calling the *ap_free*( ) function or by calling the *ap_user_dealloc*( ) function directly. • If AP_ALLOC is not set and space was available in the supplied buffers, then the AP_MORE flag is set and 0 is returned.
AP_MORE	This flag is ignored by XAP when *ap_look*( ) is called. Upon return, if all data associated with a primitive has not been received, the AP_MORE bit of the *flags* argument will be set. This indicates to the user that subsequent calls to *ap_look*( ) will return the same protocol information and user data as this call, and may as a local matter return additional user data. If the AP_MORE bit is not set, all data associated with the primitive has been received. It should be noted that the *sptype* argument must be checked after each invocation of *ap_look*( ) since an unsequenced primitive (e.g., A_PABORT_IND) may arrive and replace the original primitive. In some cases, the original primitive may be lost. Note it is possible for *ap_look*( ) to return with the AP_MORE flag set even when the AP_ALLOC flag is set, if non-blocking execution mode is being used or an expedited data primitive arrives at the XAP instance (see below).

If XAP is being used in blocking execution mode (AP_NDELAY bit of the AP_FLAGS environment attribute *is not* set), *ap_look*( ) blocks until the protocol information of a primitive (i.e. those fields in *cdata*) is available. Some or all of the user data associated with the primitive may also be returned to the user.

If XAP is being used in non-blocking execution mode (AP_NDELAY bit of the AP_FLAGS enviromnent attribute *is* set), *ap_look*( ) will return a value of –1 and the location pointed to by

aperrno_p is set to the [AP_AGAIN] error code until the protocol information of a primitive is available. Some or all of the user data associated with the primitive may also be returned to the user.

If an expedited data primitive arrives while an earlier primitive is being processed (that is, an earlier indication or confirm primitive has been examined with *ap_look*( ) or has been partly received with *ap_rcv*( )), *ap_look*( ) returns the expedited data primitive. Processing of the earlier primitive is suspended. The user should use the *ap_rcv*( ) function to retrieve the expedited data primitive. Once the expedited data primitive has been successfully received, XAP re-presents the original primitive. Note that it is possible for the original primitive to be deferred by an expedited data primitive more than once.

If an abort primitive arrives while an earlier primitive is being processed (that is, an earlier indication or confirm primitive has been examined with *ap_look*( ) or has been partly received with *ap_rcv*( )), *ap_look*( ) will return the abort primitive. The original primitive is discarded. The user should use the *ap_rcv*( ) function to retrieve the abort primitive.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

[AP_AGAIN]	XAP was unable to complete the requested action. Try again.
[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_BADFLAGS]	AP_ALLOC was set but a user-supplied buffer allocation routine was not specified on *ap_open*( ) or *ap_restore*( ).
[AP_BADLSTATE]	XAP is in a state where *ap_look*( ) is not allowed (i.e., AP_UNBOUND).
[AP_BADUBUF]	The buffers pointed to by *ubuf* are invalid.
[AP_NOBUF]	The supplied user data buffers contained no unused buffer space and, if AP_ALLOC was set, then no more buffers could be obtained.
[AP_NOENV]	There is no XAP environment associated with *fd*.
[AP_NOMEM]	Out of memory.
[AP_NOT_SUPPORTED]	*ap_look*( ) operation is not supported by this implementation of XAP.
[AP_PDUREJ]	XAP rejected the received PDU.

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

NAME

ap_open - create an XAP instance

SYNOPSIS

```
#include <xap.h>

int ap_open (
    const char *provider,
    int oflags,
    int (*ap_user_alloc) (int, ap_osi_vbuf_t **,void **,
                          int,int,unsigned long *),
    int (*ap_user_dealloc) (int, ap_osi_vbuf_t *,void *,
                            int,unsigned long *),
    unsigned long *aperrno_p )
```

DESCRIPTION

This function creates an instance of XAP using the communications provider identified by *provider*. XAP does not assign any specific interpretation to the format of this string parameter. However, individual implementations may assign additional semantics to the string in order to implement conventions applicable to a particular operating system environment.

The *oflags* argument is a bit mask used to control certain aspects of how the *ap_open*( ) invocation is handled by XAP. Legal values for the *oflags* argument are formed by OR'ing together zero or more of the flags described below.

Flag	Description
AP_NDELAY	This flag indicates the XAP instance opened is to operate in the non-blocking execution mode: AP_NDELAY will be set in the AP_FLAGS environment attribute. If this flag is not present, the opened XAP instance will operate in blocking execution mode.
AP_BUFFERS_ONLY	This flag indicates that the user-supplied memory allocation and deallocation functions are to be used only to allocate and deallocate buffers. XAPs internal memory allocation and deallocation routines will be used to allocate and deallocate environment attributes.

If AP_BUFFERS_ONLY is present in *oflags* and both *ap_user_alloc* and *ap_user_dealloc* are absent, a value of -1 is returned and the location pointed at by *aperrno_p* is set to [AP_BADFLAGS].

The *ap_user_alloc* parameter is a pointer to a user-supplied memory allocation function. The synopsis for a user-supplied memory allocation function is:

```
int ap_user_alloc(
    int fd,
    ap_osi_vbuf_t **buf,
    void **mem,
    int size,
    int type,
    unsigned long *aperrno_p )
```

This function is used by the XAP instance identified by *fd* to allocate either memory or buffer space. The *type* field takes the values AP_BUFFERS or AP_MEMORY. When AP_BUFFERS is specified, a linked set of *ap_osi_vbuf_t* structures are returned in *buf*, and *size* indicates the number of octets of space requested. Notice that *mem* is not used in this case, and that a user allocation routine may return less or more bufferspace than requested. When AP_MEMORY is specified, a block of memory is returned in *mem*, and *size* indicates the number of octets of space requested. Notice that *buf* is not used in this case. Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed to by *aperrno_p* is set to indicate the error.

The *ap_user_dealloc* parameter is a pointer to a user-supplied memory deallocation function. The synopsis for a user-supplied memory deallocation function is:

```
int ap_user_dealloc (
    int fd,
    ap_osi_vbuf_t *buf,
    void *mem,
    int type,
    unsigned long *aperrno_p )
```

This function is used by the XAP instance identified by *fd* to deallocate either the memory or buffers *that it allocated*. The *type* field takes the values AP_BUFFERS or AP_MEMORY. When AP_BUFFERS is specified, all buffers in the chain pointed to by *buf* are freed. Notice that *mem* is not used in this case. When AP_MEMORY is specified, the memory pointed to by *mem* is freed. Notice that *buf* is not used in this case. Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed to by *aperrno_p* is set to indicate the error.

The user allocation and deallocation routine parameters (*ap_user_alloc*( ) and *ap_user_dealloc*( )) must either both be present or absent. If absent, they are represented by null values. If they are absent, XAP will use built-in functions to allocate and deallocate memory from the *user memory space*. The user supplied or built-in allocation routines are called by XAP to obtain or return memory for environment attributes.

Note: XAP users are advised that allowing *ap_user_alloc* to default to the XAP-supplied memory allocation mechanism means that the service will not perform dynamic allocation of user data buffers in order to receive incoming primitives. In this case the user must either pass sufficient buffers in the call to *ap_rcv*( ) to store the user data for the incoming primitive, or the user must call *ap_rcv*( ) multiple times to receive all the user data associated with the incoming primitive (see the discussion of the AP_MORE flag in the *ap_rcv*( ) manual page).

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

On success, *ap_open*( ) returns an XAP instance identifier, an integer (≥ 0), that is used to identify the XAP instance in subsequent calls to XAP functions. Otherwise, a value of –1 is returned and the location pointed to by *aperrno_p* is set to indicate the error. (The XAP environment must be initialised using a call to *ap_init_env*( ) or *ap_restore*( ) before it can be used to send or receive primitives.)

ERRORS

[AP_BADALLOC]	The *ap_user_alloc/ap_user_dealloc* argument combination was invalid.
[AP_BADFLAGS]	The specified combination of flags is invalid.

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

NAME

ap_poll - input/output multiplexing

SYNOPSIS

```
#include <xap.h>

int
ap_poll (
    ap_pollfd_t fds[],
    int nfds,
    int timeout,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function provides users with a consistent interface for detecting when certain events have occurred on an XAP instance.

The *fds* argument is an array of *nfds* **ap_pollfd_t** structures. The **ap_pollfd_t** structure includes the following members:

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

A UNIX file descriptor is used for the purposes of clarification, although *fd* may be replaced with a comparable entity on a non-UNIX system.

The *events* field is a bitmask used to indicate which events should be reported for the instance. The *revents* field will be set by XAP to indicate which of the requested events have occurred. All XAP Library implementations recognise the following events:

AP_POLLRDNORM	Data (e.g., an XAP Library primitive or user data associated with an XAP Library primitive) has arrived on the normal data flow and is available to be read.
AP_POLLRDBAND	Data has arrived outside the normal data flow and is available to be read. This event may occur if the implementation supports out-of-band-data and a P_XDATA_IND primitive arrives. In implementations that do not support multiple data bands, expedited data will arrive on the normal data flow and this event will never occur.
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.
AP_POLLWRNORM	The same as AP_POLLOUT.
AP_POLLWRBAND	Out-of-band data can be sent. This event will never be true in implementations that do not support multiple data bands.
AP_POLLNVAL	Specified file descriptor is an invalid XAP instance identifier. This bit is only valid for *revents*.

Support for events other than those listed above is optional. Users interested in developing applications that are portable across different XAP Library implementations should keep this caveat in mind.

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. On

implementations where millisecond timing is not available, *timeout* is rounded up to the nearest legal value on the system. If the value of *timeout* is 0, *ap_poll*( ) returns immediately. 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 specified instances are operating in blocking or non-blocking execution mode.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a non-negative value is returned. A positive value 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. Upon failure, a value of –1 (FAIL) is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

Only **operating system** class errors are reported.

NAME

ap_rcv - receive an ACSE / Presentation primitive from the association / connection

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function is used to receive an indication or confirm primitive. *fd* identifies the XAP instance for which the user wishes to receive primitives.

When *ap_rcv*( ) is called, *sptype* must point to an **unsigned long**, and *cdata* must point to an **ap_cdata_t** structure.

Upon return, the value of the **unsigned long** pointed to by the *sptype* parameter will contain the symbolic constant defined in **<xap.h>** that identifies the received primitive. The symbolic constants are derived from the primitive names by prefixing the name with AP_. The table below lists the primitives that can be received using *ap_rcv*( ). The following information is provided in the table:

primitive The name of the primitive.

valid in states The states during which this primitive may be received (states are given as values of the AP_STATE attribute).

must be set A list of XAP environment attributes that *must* be set in order to be able to receive this primitive (attributes marked with † have defaults).

Note that some attributes that had to be set in order to enter a state where this primitive is legal may not be listed.

may change A list of the attributes that may change as a result of receiving this primitive.

next state The state that will be entered upon receipt of this primitive (states are given as the value of the AP_STATE attribute).

Primitive/Attribute & Primitive/State Relationships				
primitive	**valid in states**	**must be set**	**may change**	**next state**
A_ABORT_IND	all except: AP_UNBOUND AP_IDLE	none	AP_STATE	AP_IDLE
A_ASSOC_IND	AP_IDLE	AP_BIND_PADDR AP_LIB_SEL	AP_ACSE_SEL AP_CLD_AEID AP_CLD_AEQ AP_CLD_APID AP_CLD_APT AP_CLG_AEID AP_CLG_AEQ AP_CLG_APID AP_CLG_APT AP_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	AP_WASSOCrsp_ASSOCind
A_ASSOC_CNF	AP_WASSOCcnf_ASSOCreq	none	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	(AP_IDLE, AP_DATA_XFER)
A_PABORT_IND	all except: AP_UNBOUND AP_IDLE	none	AP_STATE	AP_IDLE
A_RELEASE_IND	AP_DATA_XFER AP_WRELcnf_RELreq	none	AP_STATE	AP_WRELrsp_RELind (AP_WRELrsp_RELind_init or AP_WRELcnf_RELreq_rsp)

Primitive/Attribute & Primitive/State Relationships				
primitive	**valid in states**	**must be set**	**may change**	**next state**
A_RELEASE_CNF	AP_WRELcnf_RELreq AP_WRELcnf_RELreq_rsp	none	AP_STATE	(AP_IDLE or AP_DATA_XFER) AP_WRELrsp_RELind
P_ACTEND_IND	AP_DATA_XFER	none	AP_STATE	AP_WACTErsp_ACTEind
P_ACTEND_CNF	AP_WACTEcnf_ACTEreq	none	AP_STATE	AP_DATA_XFER
P_ACTDISCARD_IND	AP_WRESYNcnf_RESYNreq AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_WRECOVERYind AP_WRECOVERYreq AP_DATA_XFER	none	AP_STATE	AP_WACTDrsp_ACTDind
P_ACTDISCARD_CNF	AP_WACTDcnf_ACTDreq	none	AP_STATE AP_TOKENS_OWNED	AP_DATA_XFER
P_ACTINTR_IND	AP_WRESYNcnf_RESYNreq AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_WRECOVERYind AP_WRECOVERYreq AP_DATA_XFER	none	AP_STATE	AP_WACTIrsp_ACTIind
P_ACTINTR_CNF	AP_WACTIcnf_ACTIreq	none	AP_STATE AP_TOKENS_OWNED	AP_DATA_XFER
P_ACTRESUME_IND	AP_DATA_XFER	none	none	no state change
P_ACTSTART_IND	AP_DATA_XFER	none	none	no state change
P_CDATA_IND	AP_DATA_XFER	none	none	AP_WCDATArsp_CDATAind
P_CDATA_CNF	AP_WCDATAcnf_CDATAreq	none	none	AP_DATA_XFER
P_CTRLGIVE_IND	AP_DATA_XFER	none	AP_TOKENS_OWNED	no state change
P_DATA_IND	AP_DATA_XFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq	none	none	no state change
P_PXREPORT_IND	AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WRECOVERYind AP_WCDATAcnf_CDATAreq AP_DATA_XFER	none	AP_STATE	AP_WRECOVERYreq (AP_WRECOVERYreq or AP_DATA_XFER) (AP_WRECOVERYreq or AP_DATA_XFER) no state change AP_WRECOVERYreq (AP_WRECOVERYreq or AP_DATA_XFER)
P_RESYNC_IND	AP_DATA_XFER AP_WRESYNcnf_RESYNreq AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WSYNCMArsp_SYNCMAind AP_WACTEcnf_ACTEreq AP_WRECOVERYind AP_WRECOVERYreq	none	AP_STATE	AP_WRESYNrsp_RESYNind

Primitive/Attribute & Primitive/State Relationships				
primitive	**valid in states**	**must be set**	**may change**	**next state**
P_RESYNC_CNF	AP_WRESYNcnf_RESYNreq	none	AP_STATE AP_TOKENS_OWNED	AP_DATA_XFER
P_SYNCMAJOR_IND	AP_DATA_XFER	none	AP_STATE	AP_WSYNCMArsp_SYNCMAind
P_SYNCMAJOR_CNF	AP_WSYNCMAcnf_SYNCMAreq	none	AP_STATE	AP_DATA_XFER
P_SYNCMINOR_IND	AP_DATA_XFER	none	none	no state change
P_SYNCMINOR_CNF	AP_DATA_XFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq	none	none	no state change
P_TDATA_IND	AP_DATA_XFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq	none	none	no state change
P_TOKENGIVE_IND	AP_DATA_XFER AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_WRECOVERYind AP_WRECOVERYreq AP_WCDATAcnf_CDATAreq	none	AP_STATE AP_TOKENS_OWNED	no state change no state change no state change no state change no state change (no state change or AP_DATA_XFER) (no state change or AP_DATA_XFER) no state change
P_TOKENPLEASE_IND	AP_DATA_XFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WCDATAcnf_CDATAreq	none	none	no state change
P_UXREPORT_IND	AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WRECOVERYind AP_DATA_XFER	none	AP_STATE	AP_WRECOVERYreq (AP_WRECOVERYreq or AP_DATA_XFER) (AP_WRECOVERYreq or AP_DATA_XFER) no state change (AP_WRECOVERYreq or AP_DATA_XFER)
P_XDATA_IND	AP_DATA_XFER AP_WRELcnf_RELreq AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq	none	none	no state change

Protocol information received with a primitive will be conveyed by the **ap_cdata_t** structure pointed to by *cdata*. The value returned in *sptype* serves as the discriminant for what members of the *cdata* are affected. A complete discussion of the use of the *cdata* parameter is provided for each XAP primitive in Chapter 7.

User-data received with a primitive will be returned to the user via the *ubuf* parameter. The XAP interface supports a vectored buffering scheme for handling user data. All data buffers are passed to XAP by the user in a chain of one or more *ap_osi_vbuf_t/ap_osi_dbuf_t* pairs. *ubuf* must point to a location holding a pointer to an **ap_osi_vbuf_t** structure, defined as follows:

```
typedef struct {
     unsigned char *db_base;      /* beginning of buffer    */
     unsigned char *db_lim;       /* last octet+1 of buffer */
     unsigned char db_ref;        /* reference count        */
} ap_osi_dbuf_t ;

typedef struct ap_osi_vbuf ap_osi_vbuf_t;
struct ap_osi_vbuf {
     ap_osi_vbuf_t *b_cont;       /* next message block     */
     unsigned char *b_rptr;       /* 1st octet of data      */
     unsigned char *b_wptr;       /* 1st free location      */
     ap_osi_dbuf_t *b_datap;      /* data block             */
} ;
```

User-data associated with XAP primitives is returned in a linked list of message blocks. Each message block is represented by an *ap_osi_vbuf_t* structure and is associated with a data block. Data blocks, which are represented by *ap_osi_dbuf_t* structures, may be associated with more than one message block. The *db_ref* field of the *ap_osi_dbuf_t* structure indicates the number of *ap_osi_vbuf_t* structures that reference a particular data block. The *db_base* and *db_lim* fields of the *ap_osi_dbuf_t* structure point to the beginning and end of the data block respectively. The *b_rptr* and *b_wptr* fields of the referencing *ap_osi_vbuf_t* structures point to the first octets to be read and written within the data block respectively. The *b_cont* field of the *ap_osi_vbuf_t* points to the next message block in the chain or is NULL if this is the end of the list.

The user allocation routine is responsible for setting up all fields of the *ap_osi_vbuf_t* and *ap_osi_dbuf_t* structures when allocating buffers. If buffers are allocated by another mechanism, the user must ensure that the fields of each *ap_osi_vbuf_t* and *ap_osi_dbuf_t* pair in the chain are set up prior to calling *ap_rcv*( ).

ap_rcv( ) places data into any buffer where write space is available (*b_wptr* < *db_lim*) and updates *b_wptr* - no other fields in the *ap_osi_vbuf_t/ap_osi_dbuf_t* structures are updated (with the exception of *b_cont* which is updated when adding further *ap_osi_vbuf_t/ap_osi_dbuf_t* pairs to the chain).

The user may pass full, partially full and empty receive buffers to *ap_rcv*( ). The user is responsible for ensuring that it is valid for the XAP library to fill any of the supplied buffers from b_wptr to db_lim.

If the user wishes all the buffers for *ap_rcv*( ) to be allocated using the user allocation routine, then the *ubuf* pointer will point to a NULL *ap_osi_vbuf_t* pointer.

The XAP user is responsible for decoding the user data received in the *ubuf* parameter. The general rules for decoding user data are stated here, please see individual manual pages in Chapter 4 for specific exceptions to these rules.

- If the ‘‘X.410-1984’’ mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the contents of *ubuf* buffer(s) are mapped directly from the SS-user data parameter of the equivalent session service primitive. Refer to the ISO Presentation Layer Protocol Specification (see reference **ISO 8823**) for further information concerning the encoding of these values. (The primary exception to this rule is the A-ASSOC primitives).
- If the ‘‘X.410-1984’’ mode of operation is *not* in effect and the primitive received is an ACSE primitive, the data received in the *ubuf* buffer(s) must be decoded according to the definition specified in the ACSE Protocol Specification (reference **ISO 8650**):

  ```
  [30] IMPLICIT SEQUENCE OF EXTERNAL
  ```

- If the ‘‘X.410-1984’’ mode of operation is *not* in effect and the primitive received is a Presentation primitive, the data received in the *ubuf* buffer(s) must be decoded according to

the User-data definition specified in the Presentation Protocol Specification (reference **ISO 8823**):

```
CHOICE {
    [APPLICATION 0] IMPLICIT OCTET STRING,
    [APPLICATION 1] IMPLICIT SEQUENCE OF PDV-list
}
```

The *flags* argument is a bit mask used to control certain aspects of XAP processing. Values for this field are formed by OR'ing together zero or more of the following flags:

Flag	Description
AP_ALLOC	If AP_ALLOC is set and the user did not specify an allocation routine on *ap_open*( ) (or *ap_restore*( )) then –1 is returned and the location pointed to by *aperrno_p* is set to the [AP_BADFLAGS] error code. If no space is available in the supplied buffer chain (or the location pointed to by *ubuf* contains NULL) and either AP_ALLOC is not set or AP_ALLOC is set but the user allocation routine refuses to supply any buffers, then the call to *ap_rcv*( ) fails, –1 is returned and the location pointed to by *aperrno_p* is set to the [AP_NOBUF] error code. Regardless of the setting of AP_ALLOC, the user must have set the location pointed to by *ubuf* either to NULL or to point to a chain of one or more buffers. The AP_ALLOC flag setting only takes effect when any supplied buffers have been filled and more data remains to be returned to the user: • If the AP_ALLOC flag is set and space was available in the supplied buffer chain, the user allocation routine is called to supply further buffers as they are needed. If the user allocation routine refuses to supply further buffers then the AP_MORE flag is set and the call to *ap_rcv*( ) completes; 0 is returned. The user must free any buffers allocated by the user allocation routine either by calling the *ap_free*( ) function or by calling the *ap_user_dealloc*( ) function directly. • If AP_ALLOC is not set and space was available in the supplied buffers, then the AP_MORE flag is set and 0 is returned.
AP_MORE	This flag is ignored by XAP when *ap_rcv*( ) is called. Upon return, if all data associated with a primitive has not been received, the AP_MORE bit of the *flags* argument will be set, and the error code [AP_AGAIN] or [AP_NOBUF] is returned. This indicates to the user that further *ap_rcv*( ) calls are required to receive the remainder of the data. If the AP_MORE bit is not set, all data associated with the primitive has been received. A primitive that is received in multiple parts may be interrupted by an incoming unsequenced primitive (P_XDATA_IND, A_ABORT_IND or A_PABORT_IND). See the discussion below for information on how this is indicated by the service and how the reception of the current primitive is resumed if appropriate. Note it is possible for *ap_rcv*( ) to return with the AP_MORE flag set even when the AP_ALLOC flag is set, if non-blocking execution mode is being used or an expedited data primitive arrives at the XAP instance (see below).

If XAP is being used in blocking execution mode (AP_NDELAY bit of the AP_FLAGS environment attribute *is not* set), *ap_rcv*( ) blocks until either an entire primitive is received, or XAP fills the buffer(s) pointed to by *ubuf*.

If XAP is being used in non-blocking execution mode (AP_NDELAY bit *is* set) and it runs out of data before receiving an entire primitive and it has not filled the ubuf buffer(s), *ap_rcv*( ) returns a value of –1 and the location pointed to by *aperrno_p* is set to the [AP_AGAIN] error code.

If an unsequenced primitive (such as expedited data or an abort primitive) arrives while another primitive is being processed, *ap_rcv*( ) returns –1 and sets the location pointed to by *aperrno_p* to the [AP_LOOK] error code. The AP_MORE bit of the flags argument is set. This indicates to the user that XAP has suspended processing of the current primitive. The user must use the *ap_rcv*( ) function to retrieve the unsequenced primitive.

For P_XDATA_IND, XAP resumes receiving data associated with the suspended primitive once the expedited data has been successfully received. If an abort primitive is received, the rest of the in-progress primitive is lost and is not returned by the service in subsequent *ap_rcv* calls.

Note that it is possible for the processing of a single primitive to be interrupted more than once due to the arrival of unsequenced indications.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

[AP_AGAIN]	XAP was unable to complete the requested action. Try again.
[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_BADFLAGS]	AP_ALLOC was set but a user-supplied buffer allocation routine was not specified on *ap_open*( ) or *ap_restore*( ).
[AP_BADLSTATE]	XAP is in a state where *ap_rcv*( ) is not allowed (i.e., AP_UNBOUND).
[AP_BADUBUF]	Either the buffers pointed to by *ubuf* are invalid, or the pointer is NULL and yet AP_ALLOC is not set.
[AP_LOOK]	An event is pending.
[AP_NOBUF]	The supplied user data buffers contained no unused buffer space and, if AP_ALLOC was set, then no more buffers could be obtained.
[AP_NOENV]	There is no XAP environment associated with *fd*.
[AP_NOMEM]	Out of memory.
[AP_PDUREJ]	XAP rejected the received PDU.

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

NAME

ap_restore - restore an XAP instance

SYNOPSIS

```
#include <stdio.h>
#include <xap.h>

int ap_restore (
    int fd,
    FILE *savef,
    int oflags,
    int (*ap_user_alloc) (int, ap_osi_vbuf_t **,void **,
                          int,int,unsigned long *),
    int (*ap_user_dealloc) (int, ap_osi_vbuf_t *,void *,
                            int,unsigned long *),
    unsigned long *aperrno_p)
```

DESCRIPTION

Used in conjunction with *ap_save*( ), this function provides a way for cooperating processes to share a single instance of XAP. The *ap_restore*( ) function recreates the XAP instance that was saved to the file associated with *savef* in the calling process's address space.

Used with a NULL *savef* argument, this function provides a means for applications which are invoked by an Association Listener (see Section 2.4 on page 32 for a description of Association Listening) to:

- establish an XAP environment
- restore the XAP instance state to AP_IDLE
- set the local local presentation address from information held within the communications provider.

The restored XAP instance will be supported by the communication endpoint identified by *fd*. This must be the same communication endpoint that supported this instance of XAP when it was saved or transferred by an association listener. As the service cannot detect whether or not a restore has been performed to the same communication endpoint, the behaviour of the service in this case is not specified.

fd will be used to identify the restored instance of XAP in subsequent invocations of XAP functions. If the communication endpoint identified by fd currently supports an XAP instance, the instance is released prior to being recreated using the snapshot saved in file *savef* along with user allocation and deallocation routines passed in the *ap_user_alloc*( ) and *ap_user_dealloc*( ) arguments.

savef must have been opened for reading. The *ap_restore*( ) function begins reading from the current position in the file and does not close *savef* when it finishes.

oflags may be used to control aspects of buffer allocation. The user may choose to supply allocation and deallocation routines for receive buffers while allowing XAP to handle allocation for memory for values of environment attributes. This is indicated by setting the AP_BUFFERS_ONLY flag of the *oflags* parameter.

The *ap_user_alloc* parameter is a pointer to a user-supplied memory allocation function. The synopsis for a user-supplied memory allocation function is:

```
int ap_user_alloc (
    int fd,
    ap_osi_vbuf_t **buf,
    void **mem,
    int size,
    int type,
    unsigned long *aperrno_p )
```

This function is used by the XAP instance identified by *fd* to allocate either memory or buffer space. The *type* field takes the values AP_BUFFERS or AP_MEMORY. When AP_BUFFERS is specified, a linked set of *ap_osi_vbuf_t* structures are returned in *buf*, and *size* indicates the number of octets of space requested. Notice that *mem* is not used in this case, and that a user allocation routine may return less or more space than requested. When AP_MEMORY is specified, a block of memory is returned in *mem*, and *size* indicates the number of octets of space requested. Notice that *buf* is not used in this case. Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed to by *aperrno_p* is set to indicate the error.

The *ap_user_dealloc* parameter is a pointer to a user-supplied memory deallocation function. The synopsis for a user-supplied memory deallocation function is:

```
int ap_user_dealloc (
    int fd,
    ap_osi_vbuf_t *buf,
    void *mem,
    int type,
    unsigned long *aperrno_p )
```

This function is used by the XAP instance identified by *fd* to deallocate either the memory or buffers *that it allocated*. The *type* field takes the values AP_BUFFERS or AP_MEMORY. When AP_BUFFERS is specified, all buffers in the chain pointed to by *buf* are freed. Notice that *mem* is not used in this case. When AP_MEMORY is specified, the memory pointed to by *mem* is freed. Notice that *buf* is not used in this case. Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed to by *aperrno_p* is set to indicate the error.

Note: If the ap_user_alloc/ap_user_dealloc argument combination is invalid, –1 is returned and the location pointed to by *aperrno_p* is set to the [AP_BADALLOC] error code; any existing XAP instance remains unchanged.

The user allocation and deallocation routine parameters (*ap_user_alloc*( ) and *ap_user_dealloc*( )) must either both be present or absent. If absent, they are represented by null values. If they are absent, XAP will use built-in functions to allocate and deallocate memory from the *user memory space*. The user supplied or built-in allocation routines are called by XAP to obtain or return memory for environment attributes.

Note: XAP users are advised that allowing *ap_user_alloc* to default to the XAP-supplied memory allocation mechanism means that the application is unable to apply flow control. In this case, special attention must be paid to the handling of unlimited user data on inbound events.

Coordination between several cooperating processes sharing the same XAP instance can be achieved by using the file permission and file and record locking capabilities of the operating system to control access to the save file.

It should be noted that the *ap_restore*( ) function does not provide a way to ‘‘roll-back’’ the state of the service provider. Events that were processed after an XAP instance was saved cannot be recovered by restoring the instance to its state before the messages were consumed.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

[AP_BADALLOC]	The *ap_user_alloc/ap_user_dealloc* argument combination was invalid.
[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_BADPARSE]	XAP was unable to read the contents of the environment from *savef*.
[AP_BADRESTR]	The environment cannot be restored because the current environment is being used to send/receive data (i.e., utilising the AP_MORE bit).
[AP_BADSAVEF]	*savef* is NULL or was opened improperly.
[AP_NOT_SUPPORTED]	The *ap_restore*( ) operation is not supported by this implementation of XAP.

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

CAVEAT

The behaviour of an XAP instance which has been restored in an improper manner is undefined. This includes all cases in which there has been *any* intervening operation on the supporting communication endpoint between the time the instance was saved and the time it was restored.

NAME

ap_save - save an XAP instance

SYNOPSIS

```
#include <stdio.h>
#include <xap.h>

int ap_save (
    int fd,
    FILE *savef,
    unsigned long *aperrno_p)
```

DESCRIPTION

Used in conjunction with *ap_restore*( ), this function provides a way for cooperating processes to share an instance of XAP. The *ap_save*( ) function writes a ''snapshot'' of XAP instance identified by *fd*, to the file identified by *savef*. Included in the saved information are the values of all XAP environment attributes and any internal state information needed to recreate this XAP instance (see *ap_restore*( ) on page 87).

savef must have been opened for writing. The *ap_save*( ) function begins writing from the current position in the file and does not close *savef* when it finishes.

The *ap_save*( ) function does not directly support any form of process coordination. However, the user can, for example, use the file permission and file and record locking capabilities of the operating system with the save file to facilitate coordination of cooperating processes sharing a single XAP instance.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_BADSAVE]	The environment cannot be saved because the current environment is being used to send/receive data (i.e., utilising the AP_MORE bit).
[AP_BADSAVEF]	*Savef* is NULL or was opened improperly.
[AP_NOENV]	There is no XAP environment associated with *fd*.
[AP_NOT_SUPPORTED]	The *ap_save*( ) operation is not supported by this implementation of XAP.

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

NAME

ap_set_env - set XAP environment attribute

SYNOPSIS

```
#include <xap.h>

int ap_set_env (
    int fd,
    unsigned long attr,
    ap_val_t val,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function changes the value of an environment attribute for the XAP instance identified by *fd*. *attr* is used to pass the symbolic constant identifying the attribute to be set as defined in the **<xap.h>** header file.

The *val* argument is a union that is used to pass the value that is to be assigned to the specified attribute. 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 the *ap_env* manual page for complete information about attribute types.

When setting the AP_BIND_PADDR attribute it should be noted that the Presentation address set will not be used until the *ap_bind*( ) function is called. Calling *ap_bind*( ) after *ap_set_env*( ) has been used to set AP_BIND_PADDR will cause the Presentation address to be validated and if the authorisation check succeeds, then the endpoint moved to the AP_IDLE state. If the *ap_set_env*( ) function is called to change AP_BIND_PADDR to a new Presentation address after a successful *ap_bind*( ), then the endpoint will not be re-bound until *ap_bind*( ) is called again.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

[AP_AGAIN]	XAP was unable to set the specified attribute at this time (e.g., because of temporary resource constraints). Try again.
[AP_BADASLSYN]	The transfer syntaxes proposed for the ACSE syntax are not supported.
[AP_BADATTRVAL]	*val* is an invalid value assignment for *attr*.
[AP_BADF]	The *fd* parameter does not identify an XAP instance.
[AP_HANGUP]	XAP state may be incorrect. Use *ap_rcv*( ) to retrieve the pending event before proceeding.
[AP_NOATTR]	*attr* is an invalid attribute type.
[AP_NOENV]	There is no XAP environment associated with *fd*.
[AP_NOMEM]	XAP could not allocate sufficient memory to store the value of the specified attribute.
[AP_NOWRITE]	The attr argument refers to an environment attribute that is not writable in the current state.

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

NAME

ap_snd - send an ACSE / Presentation primitive over the association / connection

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

This function is used to send a request or response primitive. *fd* identifies the XAP instance for which the primitive is to be sent. The *sptype* parameter contains the symbolic constant defined in **<xap.h>** that identifies the primitive to be sent. The symbolic constants are derived from the primitive names by prefixing the name with AP_. The table below lists the primitives that can be sent using *ap_snd*( ), and the associated states. The following information is provided in the table:

primitive The name of the primitive.

valid in states The states during which this primitive may be sent (states are given as values of the AP_STATE attribute).

next state The state that will be entered upon successfully issuing this primitive (states are given as the value of the AP_STATE attribute).

Primitive/State Relationships

primitive	valid in states	next state
A_ABORT_REQ	all except: AP_UNBOUND AP_IDLE	AP_IDLE
A_ASSOC_REQ	AP_IDLE	AP_WASSOCcnf_ASSOCreq
A_ASSOC_RSP	AP_WASSOCrsp_ASSOCind	(AP_IDLE, AP_DATA_XFER)
A_PABORT_REQ	all except: AP_UNBOUND AP_IDLE	AP_IDLE
A_RELEASE_REQ	AP_DATA_XFER AP_WRELrsp_RELind	AP_WRELcnf_RELreq
A_RELEASE_RSP	AP_WRELrsp_RELind AP_WRELrsp_RELind_init	(AP_IDLE or AP_DATA_XFER) AP_WRELcnf_RELreq
P_ACTDISCARD_REQ	AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WRESYNrsp_RESYNind AP_WRECOVERYreq AP_DATA_XFER	AP_WACTDcnf_ACTDreq

Primitive/State Relationships		
primitive	**valid in states**	**next state**
P_ACTDISCARD_RSP	AP_WACTDrsp_ACTDind	AP_DATA_XFER
P_ACTEND_REQ	AP_DATA_XFER	AP_WACTEcnf_ACTEreq
P_ACTEND_RSP	AP_WACTErsp_ACTEind	AP_DATA_XFER
P_ACTINTR_REQ	AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WRESYNrsp_RESYNind AP_WRECOVERYreq AP_DATA_XFER AP_WCDATAcnf_CDATAreq (if and only if no QOS extended control parameter has been selected for the association)	AP_WACTIcnf_ACTIreq
P_ACTINTR_RSP	AP_WACTIrsp_ACTIind	AP_DATA_XFER
P_ACTRESUME_REQ	AP_DATA_XFER	no state change
P_ACTSTART_REQ	AP_DATA_XFER	no state change
P_CDATA_REQ	AP_DATA_XFER	AP_WCDATAcnf_CDATAreq
P_CDATA_RSP	AP_WCDATArsp_CDATAind	AP_DATA_XFER
P_CTRLGIVE_REQ	AP_DATA_XFER	no state change
P_DATA_REQ	AP_DATA_XFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind	no state change
P_RESYNC_REQ	AP_DATA_XFER AP_WSYNCMAcnf_SYNCMAreq AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_WRESYNrsp_RESYNind AP_WRECOVERYreq	AP_WRESYNcnf_RESYNreq
P_RESYNC_RSP	AP_WRESYNrsp_RESYNind	AP_DATA_XFER
P_SYNCMAJOR_REQ	AP_DATA_XFER	AP_SYNCMAcnf_SYNCMAreq
P_SYNCMAJOR_RSP	AP_SYNCMArsp_SYNCMAind	AP_DATA_XFER
P_SYNCMINOR_REQ	AP_DATA_XFER	no state change
P_SYNCMINOR_RSP	AP_DATA_XFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind	no state change
P_TDATA_REQ	AP_DATA_XFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind	no state change
P_TOKENGIVE_REQ	AP_DATA_XFER AP_WSYNCMAcnf_SYNCMAreq AP_WACTEcnf_ACTEreq AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind	no state change no state change no state change no state change no state change

Primitive/State Relationships		
primitive	**valid in states**	**next state**
	AP_WRECOVERYreq	(no state change or AP_DATA_XFER)
P_TOKENPLEASE_REQ	AP_DATA_XFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_WCDATArsp_CDATAind	no state change
P_UXREPORT_REQ	AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind AP_DATA_XFER	AP_WRECOVERYind
P_XDATA_REQ	AP_DATA_XFER AP_WRELrsp_RELind AP_WSYNCMArsp_SYNCMAind AP_WACTErsp_ACTEind	no state change

The following table lists the environment attributes associated with each primitive. The following information is provided in the table:

primitive The name of the primitive.

must be set A list of XAP environment attributes that **must** be set prior to issuing this primitive.

Note that some attributes that had to be set in order to enter a state where this primitive is legal may not be listed. Attributes other than those listed may be required by the remote application entity.

may be used A list of XAP environment attributes may be set prior to sending this primitive and the values of which will have an affect of the primitive.

may change A list of the attributes that may change as a result of sending this primitive.

Primitive/Attribute Relationships			
primitive	**must be set**	**may be used**	**may change**
A_ABORT_REQ	none	none	AP_STATE
A_ASSOC_REQ	AP_BIND_PADDR AP_CNTX_NAME AP_LCL_PADDR AP_REM_PADDR AP_LIB_SEL	AP_ACSE_SEL AP_CLD_AEID AP_CLD_AEQ AP_CLD_APID AP_CLD_APT AP_CLG_AEID AP_CLG_AEQ AP_CLG_APID AP_CLG_APT AP_CLG_CONN_ID AP_DPCN AP_INIT_SYNC_PT AP_INIT_TOKENS AP_MODE_SEL AP_PCDL AP_PFU_SEL AP_PRES_SEL AP_QOS AP_ROLE_ALLOWED AP_SESS_SEL AP_SFU_SEL	AP_ROLE_CURRENT AP_STATE
A_ASSOC_RSP	none	AP_BIND_PADDR AP_CLD_CONN_ID AP_CNTX_NAME AP_DPCR AP_INIT_SYNC_PT AP_INIT_TOKENS AP_PCDRL AP_PRES_SEL AP_QOS AP_RSP_AEID AP_RSP_AEQ AP_RSP_APID AP_RSP_APT AP_SFU_SEL	AP_DCS AP_STATE AP_TOKENS_OWNED
A_PABORT_REQ	none	none	AP_STATE
A_RELEASE_REQ	none	none	AP_STATE
A_RELEASE_RSP	none	none	AP_STATE
P_ACTDISCARD_REQ	none	none	AP_STATE
P_ACTDISCARD_RSP	none	none	AP_STATE AP_TOKENS_OWNED
P_ACTEND_REQ	none	none	AP_STATE AP_TOKENS_OWNED
P_ACTEND_RSP	none	none	AP_STATE
P_ACTINTR_REQ	none	none	AP_STATE
P_ACTINTR_RSP	none	none	AP_STATE AP_TOKENS_OWNED

Primitive/Attribute Relationships			
primitive	**must be set**	**may be used**	**may change**
P_ACTRESUME_REQ	none	none	none
P_ACTSTART_REQ	none	none	AP_TOKENS_OWNED
P_CDATA_REQ	none	none	none
P_CDATA_RSP	none	none	none
P_CTRLGIVE_REQ	none	none	AP_TOKENS_OWNED
P_DATA_REQ	none	none	AP_TOKENS_OWNED
P_RESYNC_REQ	none	none	AP_STATE
P_RESYNC_RSP	none	none	AP_STATE AP_TOKENS_OWNED
P_SYNCMAJOR_REQ	none	none	AP_STATE AP_TOKENS_OWNED
P_SYNCMAJOR_RSP	none	none	AP_STATE
P_SYNCMINOR_REQ	none	none	AP_TOKENS_OWNED
P_SYNCMINOR_RSP	none	none	none
P_TDATA_REQ	none	none	none
P_TOKENGIVE_REQ	none	none	AP_STATE AP_TOKENS_OWNED
P_TOKENPLEASE_REQ	none	none	none
P_UXREPORT_REQ	none	none	AP_STATE
P_XDATA_REQ	none	none	none

ap_snd( ) returns [AP_BADLSTATE] when *sptype* indicates a primitive which is not valid in the current state of the XAP instance. This error indicates a program logic error. Thus the XAP-user must keep track of the state of the instance.

ap_snd( ) returns [AP_LOOK] when:

1. The primitive specified by *sptype* is made invalid by an incoming event which has been processed by the underlying service provider but which has not yet been received by the XAP-user,

   or

2. expedited data is available for the XAP-user to receive.

The [AP_LOOK] return code does not indicate a program logic error. It only indicates that the XAP-user should issue *ap_rcv*( ) calls to process one or more outstanding incoming events and then take action appropriate to the current state of the instance. For example, since receiving expedited data does not result in a state change, the *ap_snd*( ) which returned [AP_LOOK] could be reissued. This includes an *ap_snd*( ) that is part of an in-progress send discussed below.

As another example, suppose XAP receives a P-RESYNC indication primitive in state AP_DATA_XFER. The XAP state is inconsistent with that of the presentation service provider. If the XAP-user issues *ap_snd*( ) to send a P_DATA_REQ primitive, XAP returns [AP_LOOK] forcing the user to call *ap_rcv*( ) to receive the P_RESYNC_IND primitive. This causes the XAP instance's state to become AP_WRESYNrsp_RESYNind. The XAP-user should send a P_RESYNC_RSP primitive to return to state AP_DATA_XFER.

A P_RESYNC_IND primitive has the effect of terminating any send in progress at that point. An XAP-user, after getting [AP_LOOK] and receiving P_RESYNC_IND, should assume any in-progress send was terminated by XAP.

Note: [AP_LOOK] implies that the XAP implementation includes some mechanism which permits a delay between a primitive being processed by the service provider and that primitive being passed to the API user. Thus, some implementations of XAP may not be capable of generating this return code.

If the primitive being sent is to be accompanied by protocol information, that information must be contained in an **ap_cdata_t** structure pointed to by *cdata*. The *man-pages* in Chapter 7 describe the use of the *cdata* argument with each XAP primitive. If no additional protocol information is to be sent with an XAP primitive, *cdata* may be NULL.

User-data can be sent with many XAP primitives. If no user-data is to be sent with a primitive, *ubuf* may be set to NULL. To send data, *ubuf→buf* must point to a linked list of *ap_osi_vbuf_t* structures. These structures allow data stored in several different buffers to be sent with a single *ap_snd*( ) invocation. The *ap_osi_vbuf_t* structure is defined as shown below.

```
typedef struct {
     unsigned char *db_base;      /* beginning of buffer    */
     unsigned char *db_lim;       /* last octet+1 of buffer */
     unsigned char  db_ref;       /* reference count        */
} ap_osi_dbuf_t ;

typedef struct ap_osi_vbuf ap_osi_vbuf_t;
struct ap_osi_vbuf {
     ap_osi_vbuf_t *b_cont;           /* next message block */
     unsigned char *b_rptr;           /* 1st octet of data  */
     unsigned char *b_wptr;           /* 1st free location  */
     ap_osi_dbuf_t *b_datap;          /* data block         */
} ;
```

The *b_cont* field of the *ap_osi_vbuf_t* structure points to the next buffer in the chain or is NULL if this is the end of the list. The *b_datap* element points to a data block that contains encoded user data. The *b_rptr* element points to the beginning of the user-data within the data block while *b_wptr* references the location following the last octet of data in the buffer.

Each data block is represented by an *ap_osi_dbuf_t* structure. The *db_ref* element of the *ap_osi_dbuf_t* structure indicates the number of *ap_osi_vbuf_t* structures that reference this data block. The *db_base* element points to the beginning of a buffer and *db_lim* indicates the end of that buffer (buffer size == *db_lim-db_base*).

The XAP user is responsible for encoding the user data passed to XAP in the *ubuf* parameter. The general rules for encoding user data are stated here; please see individual manual pages in Chapter 7, for specific exceptions to these rules.

- If the ''X.410-1984'' mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the contents of *ubuf* buffer(s) are mapped directly to the SS-user data parameter of the equivalent session service primitive. Refer to the ISO Presentation Layer Protocol Specification (reference **ISO 8823**) for further information concerning the encoding of these values. (The primary exception to this rule is the A_ASSOC_REQ and A_ASSOC_RSP primitives).
- If the ''X.410-1984'' mode of operation *is not* in effect and the primitive to be sent is an ACSE primitive, the data in the *ubuf* buffer(s) must be encoded according to the definition specified in the ACSE Protocol Specification (reference **ISO 8650**):

```
[30] IMPLICIT SEQUENCE OF EXTERNAL
```

- If the ‘‘X.410-1984’’ mode of operation *is not* in effect and the primitive to be sent is a Presentation primitive, the data in the *ubuf* buffer(s) must be encoded according to the User-data definition specified in the Presentation Protocol Specification (reference **ISO 8823**):

```
CHOICE {
    [APPLICATION 0] IMPLICIT OCTET STRING,
    [APPLICATION 1] IMPLICIT SEQUENCE OF PDV-list
}
```

The *flags* argument is a bit mask that can be used to control certain aspects of how the *ap_snd*( ) invocation is handled by XAP. Legal values for the *flags* argument are formed by OR’ing together zero or more of the flags described below.

Flag	Description
AP_MORE	This flag indicates that data associated with the specified primitive will be sent with multiple *ap_snd*( ) calls. Each *ap_snd*( ) call with the AP_MORE bit set indicates that another *ap_snd*( ) will follow with additional data associated with the specified primitive. The value of the *sptype* argument must be the same for all *ap_snd*( ) calls used to send a single primitive. Calling *ap_snd*( ) with the AP_MORE bit reset signals that the primitive is complete.
AP_QUERY_DATA_PENDING	This flag indicates to XAP that a check shall be made for the availability of incoming data on the connection and that if data is available this shall be indicated by returning a result of –1 and setting *aperrno_p* as indicated below.

If XAP is being used in blocking execution mode (i.e., the AP_NDELAY bit of the AP_FLAGS attribute *is not* set), *ap_snd*( ) blocks until sufficient resources are available to permit all of the data in the *ubuf* buffer(s) to be sent. If XAP is being used in non-blocking execution mode (i.e., the AP_NDELAY bit of the AP_FLAGS attribute is set), *ap_snd*( ) may return after having sent only a portion of the data to the A / P-Provider. If all data is not sent, *ap_snd*( ) will return a value of –1 and the location pointed to by *aperror_p* is set to the [AP_AGAIN] error code. The user must continue to call *ap_snd*( ) with exactly the same arguments until the function completes successfully (i.e., returns a value of 0).

If AP_MORE is set by the user or if [AP_AGAIN] is returned by XAP, sending a primitive requires multiple invocations of *ap_snd*( ). In general, *ap_snd*( ) is issued repeatedly with the same primitive until:

1. The user resets the AP_MORE flag

   and

2. XAP returns success, i.e., does not return the [AP_AGAIN] error code

   or

3. XAP returns the [AP_LOOK] or [AP_HANGUP] error codes.

An association can be aborted by the user even if a send is ‘‘in progress’’, i.e., conditions 1 and 2 have not been met. An *ap_snd*( ) specifying A_ABORT_REQ or A_PABORT_REQ will cause the in-progress send and the association to be aborted. An *ap_close*( ) will also have this effect.

It is not permissible to issue *ap_snd*( ) specifying any primitive other than A_ABORT_REQ or A_PABORT_REQ while there is a *send* in progress. If this is attempted, XAP returns the [AP_BADLSTATE] error code.

The XAP user must not prematurely terminate an in-progress send by resetting AP_MORE as this will result in a partial APDU being sent to the remote system which, in turn, may cause the remote system to abort the application association.

aperrno_p must be set to point to a location which will be used to carry an error code back to the user.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and the location pointed at by *aperrno_p* is set to indicate the error.

ERRORS

[AP_ACCES] The user is not authorised to use the address specified for AP_BIND_PADDR.

[AP_AGAIN] XAP was unable to complete the requested action. Try again.

[AP_AGAIN_DATA_PENDING]
XAP was unable to complete the requested action. Try again. There is an event available for the user to receive.

[AP_BADDATA] User data not allowed on this service.

[AP_BADENV] A mandatory environment attribute is not set.

[AP_BADF] The *fd* parameter does not identify an XAP instance.

[AP_BADFLAGS] The specified combination of flags is invalid.

[AP_BADLSTATE] The specified primitive cannot be issued in current state.

[AP_BADPRIM] The specified primitive is not valid (i.e., unknown type, or known type but corresponds to an unavailable service).

[AP_BADUBUF] The length given for user data does not match what was sent; or the AP_MORE bit was reset but no data was given for a primitive that is not associated with either an ACSE or Presentation PDU.

[AP_DATA_OVERFLOW]
User data and presentation service pci exceeds 512 bytes on session V1 or the length of the user data exceeds a locally defined limit, as stated in the CSQ.

[AP_HANGUP] The association has been aborted. Use *ap_rcv*( ) to read the abort indication.

[AP_LOOK] A pending event requires attention.

[AP_NOENV] There is no XAP environment associated with *fd*.

[AP_SUCCESS_DATA_PENDING]
The requested action was completed successfully. There is an event available for the user to receive.

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

Chapter 5

XAP Commands

This chapter describes the XAP commands, of which there is only one - ap_osic.
The command, including its usage, is described in *manual page* format.
Support for ap_osic is optional.

NAME

ap_osic - XAP Library OSI information compiler

SYNOPSIS

```
ap_osic [ options ] files
```

DESCRIPTION

The *ap_osic* command processes **ap_env_file** files to generate an environment initialisation file that can be used by the *ap_init_env*( ) function to initialise the XAP Library environment. The *ap_osic* command is optional; implementations required to be portable cannot rely on it being available on all platforms.

The format of the **ap_env_file** input files is defined in Section 6.1 on page 105.

One or more **ap_env_file** files can be named on the command line. The *ap_osic* command parses these files, checks them for errors, and writes the combined initialisation information to a file named **ap_osi.env**. The following options are interpreted by *ap_osic*:

-o outfile
Write output to **outfile** instead of **ap_osi.env**.

-v By default, the attributes named in each **ap_env_file** file are assigned values in a specific order regardless of the order that they appear in that file. This is to prevent the case where attribute A is assigned a value before attribute B when the value of B may affect the allowable values for A. The user may override this default ordering by specifying the **-v** option. If this option is used, environment attributes will be assigned values in the same order that they appear in the **ap_env_file** file or files.

The default attribute assignment order used in the absence of the **-v** option is:

```
AP_LIB_SEL, AP_ACSE_SEL, AP_ROLE_ALLOWED, AP_CNTX_NAME,
AP_SESS_SEL, AP_BIND_PADDR,
AP_CLD_AEID, AP_CLD_AEQ, AP_CLD_APID, AP_CLD_APT,
AP_CLG_AEID, AP_CLG_AEQ, AP_CLG_APID, AP_CLG_APT,
AP_DPCN, AP_MODE_SEL, AP_PCDL, AP_PFU_SEL,
AP_PRES_SEL, AP_REM_PADDR,
AP_RSP_AEID, AP_RSP_AEQ, AP_RSP_APID, AP_RSP_APT,
AP_INIT_SYNC_PT, AP_SFU_SEL, AP_INIT_TOKENS, AP_FLAGS,
AP_CLG_CONN_ID, AP_CLD_CONN_ID, AP_OLD_CONN_ID,
AP_AFU_SEL, AP_COPYENV, AP_DPCR, AP_QLEN, AP_QOS.
```

FILES

```
ap_osi.env      default output file
```

CAVEAT

The output from the *ap_osic* command of one XAP implementation is not necessarily readable by the *ap_init_env*( ) function of another XAP implementation, as the format of the intermediate file is not defined. Environment initialisation files are therefore only guaranteed to be portable in the **ap_env_file** form.

DIAGNOSTICS

Most diagnostic messages produced by *ap_osic* begin with the line number and name of the file in which the error was detected. If one of these conditions is detected, no output is written to the output file. The following error messages may occur:

Cannot open file	One of the specified input files cannot be opened for reading.
Syntax error	There is a syntax error in the ap_env_file file. Refer to the **ap_env_file** manual page for a description of the proper syntax.

Illegal attribute	An illegal attribute name was specified in the **ap_env_file** file.
Illegal value	An illegal value was assigned to an attribute in the **ap_env_file** file. In addition to these errors, the *ap_osic* command produces a warning (below) if multiple assignments to a single attribute are encountered. In this case, only the first assignment is used and a warning is written to *stderr* for each additional initialisation value encountered for that attribute. If no errors are detected, output will be written to the output file.
Duplicate attribute ignored	More than one assignment was encountered for a single attribute. The first value is used.

Chapter 6

XAP File Formats

This chapter defines the format of files used by XAP.

6.1 Environment File

This defines the format of an ACSE / Presentation Library initialisation file.

An **ap_env_file** is an ASCII file containing a list of XAP environment variable assignments. It is used as input to the *ap_osic* command, which generates a compiled version of the assignments for use as an environment initialisation file by the *ap_init_env*( ) function. Support of the *ap_osic* command and intialisation of the XAP environment from an input file is optional, so this mechanism may not be available in all implementations of XAP.

Each **ap_env_file** file consists of entries of the following types:

- Assignment pairs of the form:

  ```
  <attribute name> = <value>
  ```

  where `<attribute name>` is the name of an XAP library environment attribute (see Chapter 3) and `<value>` is a legal value for the attribute.

- C-style comments (/*....*/) with the syntax and semantics defined by ISO C.

- #include lines with the syntax and semantics defined by the ISO C preprocessor.

- #define lines with the syntax and semantics defined by the ISO C preprocessor for the ''#define identifier token-sequence'' form.

An entry may be split across multiple lines by terminating intermediate lines with a backslash character (\). Otherwise each entry must occupy a single line.

An **ap_env_file** file may contain assignments for any of the settable XAP Library environment attributes. The assignment pairs may appear in any order provided each pair begins on a new line.

Since not all attributes are of the same type, the format of `<value>` depends upon the particular attribute being initialised. Table 6-1 lists all of the attributes that may be initialised in an **ap_env_file** file and the format each requires for the `<value>` component of its initialisation pair.

attribute name	type	value format
AP_ACSE_SEL	unsigned long	bitmask
AP_AFU_SEL	unsigned long	bitmask
AP_BIND_PADDR	ap_paddr_t	address
AP_CLD_AEID	ap_aei_api_id_t	encoded string
AP_CLD_AEQ	ap_aeq_t	encoded string
AP_CLD_APID	ap_aei_api_id_t	encoded string
AP_CLD_APT	ap_apt_t	encoded string
AP_CLD_CONN_ID	ap_conn_id_t	connection identifier
AP_CLG_AEID	ap_aei_api_id_t	encoded string
AP_CLG_AEQ	ap_aeq_t	encoded string
AP_CLG_APID	ap_aei_api_id_t	encoded string
AP_CLG_APT	ap_apt_t	encoded string
AP_CLG_CONN_ID	ap_conn_id_t	connection identifier
AP_CNTX_NAME	ap_objid_t	object identifier
AP_COPYENV	long	integer constant
AP_DPCN	ap_dcn_t	default context name
AP_FLAGS	unsigned long	bitmask
AP_INIT_SYNC_PT	unsigned long	integer constant
AP_INIT_TOKENS	unsigned long	bitmask
AP_LIB_SEL	unsigned long	bitmask
AP_MODE_SEL	long	integer constant
AP_PCDL	ap_cdl_t	context definition list
AP_PFU_SEL	unsigned long	bitmask
AP_PRES_SEL	unsigned long	bitmask
AP_QLEN	long	integer constant
AP_QOS	ap_qos_t	quality of service
AP_REM_PADDR	ap_paddr_t	address
AP_ROLE_ALLOWED	unsigned long	bitmask
AP_RSP_AEID	ap_aei_api_id_t	encoded string
AP_RSP_AEQ	ap_aeq_t	encoded string
AP_RSP_APID	ap_aei_api_id_t	encoded string
AP_RSP_APT	ap_apt_t	encoded string
AP_SESS_SEL	unsigned long	bitmask
AP_SFU_SEL	unsigned long	bitmask

Table 6-1 Attributes that may be Initialised in an Environment File

Below is a description of the <*value*> formats specified in the preceding table. Note that blanks, newlines, horizontal and vertical tabs and form feeds in the **ap_env_file** file are considered *white space* and are ignored except as token separators.

Address

Values in this format must be given as

```
{[p_selector], [s_selector], [t_selector], {{n_address} [,{n_address}]*}}
```

where p_selector, s_selector, t_selector, and n_address are defined as follows:

p_selector: A value in the octet string format of any length.

s_selector: A value in the octet string format whose length cannot exceed 16 octets.

t_selector: A value in the octet string format whose length cannot exceed 32 octets.

n_address: One or more network addresses may be specified here in a comma separated list. Each network address is represented by a value in the octet string format whose length is less than or equal to 20, followed by an integer constant declaring the associated network type. When multiple network address components are included in a presentation address, the specific network address(es) chosen by the provider and the manner by which it is selected for initiating or listening to connections is not specified by XAP and is a local implementation issue.

Examples:

```
AP_BIND_PADDR = {, {01}, {0F}, {{{4901},AP_CLNS}}}
AP_REM_PADDR = {{01}, , , {{{4901},AP_CLNS}, \
                           {{4902},AP_UNKNOWN}}}
```

Bitmask

Values in this format must be given as one or more items in the integer constant format OR'ed together.

Examples:

```
AP_ACSE_SEL =  0x1
AP_SESS_SEL =  01 |  02
AP_SFU_SEL = AP_SESS_DUPLEX | AP_SESS_RESYNC
```

Connection Identifier

Values in this format must be given as a sequence of 3 values in the octet string format enclosed in braces and separated by commas. The 3 elements of this sequence correspond, in order, to the Calling (or Called) SS-user reference, Common reference, and Additional reference information components of the session connection identifier parameter.

Example:

```
AP_CLG_CONN_ID = { \
    {"MyCallingSS-userReference"}, \
    {"CommonRef"}, \
    {} }
```

Context Definition List

Values in this format must be given as a sequence of comma-separated 3-tuples enclosed in braces. Each 3- tuple comprises the following comma-separated elements:

1. a presentation context identifier in integer constant format
2. an abstract syntax name in object identifier format
3. a braced, comma separated sequence of transfer syntax names where each transfer syntax name is in object identifier format.

Example:

```
AP_PCDL = {\
    {1,\
        {joint_iso_ccitt 2 2 0 1},\
        {{joint_iso_ccitt asn1 basic_encoding}}\
    },\
    {3,\
        {iso 3 9999 100 2 1 1},\
        {{joint_iso_ccitt asn1 basic_encoding},\
            {iso 3 9999 100 6 1 1}}\
    }\
}
```

Note that the above identifiers of OBJECT IDENTIFIER component values use an ''_'' (underscore) character as a separator instead of a ''-'' (hyphen). For example, joint-iso-ccitt is defined as joint_iso_ccitt. Since the environment file format is ''C'' structure based, using a ''-'' as a separator would constitute an expression and not a definition.

Default Context Name

Values in this format must be given as a 2-tuple enclosed in braces. The first element of the 2-tuple is an abstract syntax name; the second is a transfer syntax name. Both elements must be in the object identifier format and the two elements are separated by a comma.

Example:

```
AP_DPCN = {{2 2 1 0 1}, {2 1 1}}
```

Integer Constant

Values in this format must be given as one of the following:

- a decimal integer
- an octal integer (prefixed by 0)
- a hexadecimal integer (prefixed by 0x or 0X)
- a symbolic constant that is either defined by the user in the **ap_env_file** file (using #define), or defined in a file included in the *ap_env_file* file (using #include).

Note that the constants in the **<xap.h>** header file are included automatically. Users are cautioned against redefining any of the constants in that file.

Examples:

```
AP_ROLE_ALLOWED = AP_RESPONDER
AP_CLD_AEID = 0xA2
```

Object Identifier

Values in this format must be given as a sequence of values in the integer constant format that are separated by blanks and enclosed in braces.

The following identifiers of OBJECT IDENTIFIER component values have been assigned by ISO and CCITT and are recognised by ap_osic:

```
iso, standard, registration_authority, member_body,
identified_organisation, ccitt, recommendation,
question, administration, network_operator,
joint_iso_ccitt, asn1, basic_encoding.
```

Note that the above identifiers use an ''_'' (underscore) character as a separator instead of a ''-'' (hyphen). For example, joint-iso-ccitt is defined as joint_iso_ccitt. Since the environment file format is ''C'' structure based, using a ''-'' as a separator would constitute an expression and not a definition.

In addition, the user may define other identifier values by using the #define preprocessor construct.

Examples:

```
AP_CNTX_NAME = {iso standard 8571 1}
AP_CNTX_NAME = {1 0 8571 1}
```

Octet String

Values in this format must be given as either an even number of hexadecimal digits or a legal C language string constant enclosed in braces. Characters in string constants will be treated as 8-bit values where bit 8 (MSB) is 0 and the low order 7 bits correspond to the character's ASCII encoding.

Examples:

```
octetstring = {000ff0ff}
octetstring = {"my string"}
```

Encoded String

Values in this format must be given as a single value in the octet string format. This octet string must correspond to a valid encoding of an ASN.1 type value.

Examples:

```
AP_CLD_APT = {06062B80CE0F0107}
```

Quality of Service

Values in this format must be given as

```
{ {throughput}, {transdel}, {reserrorrate},
  {transfailprob}, {estfailprob}, {relfailprob},
  {estdelay}, {reldelay}, {connresil},
  protection, priority, optimisedtrans, extcntl
}
```

where throughput and transdel must each be given as a pair `{maximum}`, `{average}`; `maximum` and `average` must each be given as a pair `{called}`, `{calling}`; `called` and `calling` must each be given as a pair of numeric values: `target`, `minimumacceptable` (see examples below).

reserrorrate, *transffailprob*, *estfailprob*, *relfailprob*, *estdelay*, *reldelay* and *connresil* each consisting of a pair of numeric values, the first being the target value and the second being the minimum acceptable value.

protection, *priority*, *optimisedtrans* and *extcntl* are integer constants.

Example:

```
AP_QOS = {\
/* throughput */\
{\
    /* maxthrpt */\
    {\
        /* called */\
        { /* targetvalue */ -1, /* minacceptvalue */ -1 },\
        /* calling */\
        { /* targetvalue */ -1, /* minacceptvalue */ -1 }\
    },\
    /* avgthrpt */\
    {\
        /* called */\
        { /* targetvalue */ -1, /* minacceptvalue */ -1 },\
        /* calling */\
        { /* targetvalue */ -1, /* minacceptvalue */ -1 }\
    }\
},\
/* transdel */\
{\
    /* maxdel */\
    {\
        /* called */\
        { /* targetvalue */ -1, /* minacceptvalue */ -1 },\
        /* calling */\
        { /* targetvalue */ -1, /* minacceptvalue */ -1 }\
    },\
    /* avgdel */\
    {\
        /* called */\
        { /* targetvalue */ -1, /* minacceptvalue */ -1 },\
        /* calling */\
        { /* targetvalue */ -1, /* minacceptvalue */ -1 }\
    }\
},\
/* reserrorrate */\
{ /* targetvalue */ -1, /* minacceptvalue */ -1 },\
/* transffailprob */\
{ /* targetvalue */ -1, /* minacceptvalue */ -1 },\
/* estfailprob */\
{ /* targetvalue */ -1, /* minacceptvalue */ -1 },\
/* relfailprob */\
{ /* targetvalue */ -1, /* minacceptvalue */ -1 },\
/* estdelay */\
{ /* targetvalue */ -1, /* minacceptvalue */ -1 },\
/* reldelay  */\
{ /* targetvalue */ -1, /* minacceptvalue */ -1 },\
/* connresil  */\
{ /* targetvalue */ -1, /* minacceptvalue */ -1 },\
/* protection */\
0,\
/* priority */\
AP_PRIDFLT,\
/* optimisedtrans */\
AP_NO,\
/* extcntl */\
AP_YES\
}
```

Chapter 7

XAP Primitives

This chapter presents *manual pages* for each of the primitives of the underlying OSI services to which the XAP provides access via the *ap_snd*( ) and *ap_rcv*( ) functions.

Each *man-page* provides a short description of an ACSE or Presentation Layer primitive, including the circumstances under which it may be sent or received, followed by a detailed description of the parameters associated with it.

NAME

A_ABORT_REQ - used to abort an association

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the A_ABORT_REQ primitive and restrictions on its use.

To send an A_ABORT_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_A_ABORT_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;  /* length of user-information */
                    /* field of APDU              */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

A_ABORT_IND - used to indicate an abort request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the A_ABORT_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

sptype The **unsigned long** pointed to by this argument will be set to AP_A_ABORT_IND.

cdata The following members of *cdata* are used for this primitive:

```
long src;           /* source of abort */
```

cdata→src will be set to indicate the source of the abort request. The possible values for *cdata→src* when the ‘‘normal’’ mode of operation is in effect are given below.

AP_ACSE_USER abort requested by ACSE user.

AP_ACSE_PROV abort requested by ACSE provider.

If the ‘‘X.410-1984’’ mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→src* argument will be set to AP_SRC_NOVAL by the library.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

A_ASSOC_REQ - used to initiate an association

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The A_ASSOC_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library to initiate the establishment of an association between two application entities. After sending an A_ASSOC_REQ primitive, no other primitives can be issued, except A_ABORT_REQ or A_PABORT_REQ, until A_ASSOC_CNF primitive is received.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the A_ASSOC_REQ primitive and restrictions on its use.

To send an A_ASSOC_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_A_ASSOC_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-information   */
                        /* field                        */
ap_a_assoc_env_t *env; /* environment attribute values */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

The *cdata→env* argument can be used to override XAP environment attribute values used as parameters to the A-ASSOCIATE request service. If no attribute values are to be overridden, *cdata→env* may be set to NULL. Otherwise, *cdata→env* must point to an **ap_a_assoc_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask        */
unsigned long mode_sel;      /* AP_MODE_SEL     */
ap_objid_t cntx_name;        /* AP_CNTX_NAME    */
ap_aei_api_id_t clg_aeid;    /* AP_CLG_AEID     */
ap_aeq_t clg_aeq;            /* AP_CLG_AEQ      */
ap_aei_api_id_t clg_apid;    /* AP_CLG_APID     */
ap_apt_t clg_apt;            /* AP_CLG_APT      */
ap_aei_api_id_t cld_aeid;    /* AP_CLD_AEID     */
ap_aeq_t cld_aeq;            /* AP_CLD_AEQ      */
ap_aei_api_id_t cld_apid;    /* AP_CLD_APID     */
ap_apt_t cld_apt;            /* AP_CLD_APT      */
ap_paddr_t rem_paddr;        /* AP_REM_PADDR    */
ap_cdl_t pcdl;               /* AP_PCDL         */
ap_dcn_t dpcn;               /* AP_DPCN         */
ap_qos_t qos;                /* AP_QOS          */
unsigned long a_version_sel; /* AP_ACSE_SEL     */
unsigned long p_version_sel; /* AP_PRES_SEL     */
unsigned long s_version_sel; /* AP_SESS_SEL     */
unsigned long afu_sel;       /* AP_AFU_SEL      */
unsigned long pfu_sel;       /* AP_PFU_SEL      */
unsigned long sfu_sel;       /* AP_SFU_SEL      */
ap_conn_id_t *clg_conn_id;   /* AP_CLG_CONN_ID  */
unsigned long init_sync_pt;  /* AP_INIT_SYNC_PT */
unsigned long init_tokens;   /* AP_INIT_TOKENS  */
```

The *mask* element of this structure is a bit mask indicating which environment attributes associated with this primitive are to be overridden. Values for this field are formed by OR'ing together zero or more of the flags listed in the table below. When a bit is set, the value of the associated parameter will be taken from the *cdata* argument rather than from the XAP environment. Specifying a value for a particular parameter in the *cdata* argument has the same effect on the value of the corresponding attribute in the XAP environment as calling *ap_set_env*( ). See the description of the related environment attributes for information on how the fields of this argument are used.

flag	parameter	field
AP_A_VERSION_SEL_BIT	ACSE Version Selector	*a_version_sel*
AP_AFU_SEL_BIT	ACSE Requirements	*afu_sel*
AP_CLD_AEID_BIT	Called AE Invocation-identifier	*cld_aeid*
AP_CLD_AEQ_BIT	Called AE Qualifier	*cld_aeq*
AP_CLD_APID_BIT	Called AP Invocation-identifier	*cld_apid*
AP_CLD_APT_BIT	Called AP Title	*cld_apt*
AP_CLG_AEID_BIT	Calling AE Invocation-identifier	*clg_aeid*
AP_CLG_AEQ_BIT	Calling AE Qualifier	*clg_aeq*
AP_CLG_APID_BIT	Calling AP Invocation-identifier	*clg_apid*
AP_CLG_APT_BIT	Calling AP Title	*clg_apt*
AP_CLG_CONN_ID_BIT	Session-connection Identifier	*clg_conn_id*
AP_CNTX_NAME_BIT	Application Context Name	*cntx_name*
AP_DPCN_BIT	Default Presentation Context Name	*dpcn*

flag	parameter	field
AP_INIT_SYNC_PT_BIT	Initial Synchronization Serial Point Number	*init_sync_pt*
AP_INIT_TOKENS_BIT	Initial Token Assignment	*init_tokens*
AP_MODE_SEL_BIT	Mode	*mode_sel*
AP_P_VERSION_SEL_BIT	Presentation Version Selector	*p_version_sel*
AP_PCDL_BIT	Presentation Context Definition List	*pcdl*
AP_PFU_SEL_BIT	Presentation Requirements	*pfu_sel*
AP_QOS_BIT	Quality of Service	*qos*
AP_REM_PADDR_BIT	Called Presentation Address	*rem_paddr*
AP_S_VERSION_SEL_BIT	Session Version Selector	*s_version_sel*
AP_SFU_SEL_BIT	Session Requirements	*sfu_sel*

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

If the X.410-1984 mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the contents of *ubuf* buffer(s) is assumed to be a value of type SET encoded according to the Basic Encoding Rules for ASN.1. The SET value should have an implicit context-specific tag with a value of 1 (i.e., [1] IMPLICIT). However, the XAP Library will not examine this value to verify that it is valid. Refer to the ISO Presentation Layer protocol definition (reference **ISO 8823**) for further information concerning the encoding of these values.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADROLE] The AP_INITIATOR bit of the AP_ROLE_ALLOWED attribute is not set.

NAME

A_ASSOC_IND - used to indicate a request to establish an association

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the A_ASSOC_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_A_ASSOC_IND.

cdata The following members of *cdata* are used for this primitive:

```
ap_a_assoc_env_t *env;  /*environment attribute values*/
```

The *cdata→env* argument can be used to retrieve the values of the XAP environment attributes that correspond to parameters of the A-ASSOCIATE indication service. If the AP_COPYENV attribute in the XAP environment is *false*, these values will not be returned in the *cdata* argument and *cdata→env* will be set to *null* when *ap_rcv*( ) returns. If AP_COPYENV is *true*, the XAP library will allocate an **ap_a_assoc_env_t** structure and any necessary substructures and return a pointer to it in *cdata→env*. The caller can release the storage allocated for the **ap_a_assoc_env_t** structure and its substructures by passing a pointer to *cdata* to *ap_free*( ). The following elements of the **ap_a_assoc_env_t** structure are used for this primitive:

```
unsigned long mask;          /* bit mask        */
unsigned long mode_sel;      /* AP_MODE_SEL     */
ap_objid_t cntx_name;        /* AP_CNTX_NAME    */
ap_aei_api_id_t clg_aeid;    /* AP_CLG_AEID     */
ap_aeq_t clg_aeq;            /* AP_CLG_AEQ      */
ap_aei_api_id_t clg_apid;    /* AP_CLG_APID     */
ap_apt_t clg_apt;            /* AP_CLG_APT      */
ap_aei_api_id_t cld_aeid;    /* AP_CLD_AEID     */
ap_aeq_t cld_aeq;            /* AP_CLD_AEQ      */
ap_aei_api_id_t cld_apid;    /* AP_CLD_APID     */
ap_apt_t cld_apt;            /* AP_CLD_APT      */
ap_paddr_t rem_paddr;        /* AP_REM_PADDR    */
ap_cdl_t pcdl;               /* AP_PCDL         */
ap_cdrl_t pcdrl;             /* AP_PCDRL        */
ap_dcn_t dpcn;               /* AP_DPCN         */
ap_qos_t qos;                /* AP_QOS          */
unsigned long a_version_sel; /* AP_ACSE_SEL     */
unsigned long p_version_sel; /* AP_PRES_SEL     */
unsigned long s_version_sel; /* AP_SESS_SEL     */
unsigned long afu_sel;       /* AP_AFU_SEL      */
unsigned long pfu_sel;       /* AP_PFU_SEL      */
unsigned long sfu_sel;       /* AP_SFU_SEL      */
ap_conn_id_t *clg_conn_id;   /* AP_CLG_CONN_ID  */
unsigned long init_sync_pt;  /* AP_INIT_SYNC_PT */
unsigned long init_tokens;   /* AP_INIT_TOKENS  */
```

The *mask* element of this structure is a bit mask indicating which parameters associated with this primitive were received. Values for this field are formed by OR'ing together zero or more of the flags listed in the table below. When a bit is set, the specified parameter was received. Otherwise, the parameter was not received and the corresponding field in the **ap_a_assoc_env_t** structure is not set.

flag	parameter	field
AP_MODE_SEL_BIT	Mode	*mode_sel*
AP_CNTX_NAME_BIT	Application Context Name	*cntx_name*
AP_CLG_AEID_BIT	Calling AE Invocation-identifier	*clg_aeid*
AP_CLG_AEQ_BIT	Calling AE Qualifier	*clg_aeq*
AP_CLG_APID_BIT	Calling AP Invocation-identifier	*clg_apid*
AP_CLG_APT_BIT	Calling AP Title	*clg_apt*
AP_CLD_AEID_BIT	Called AE Invocation-identifier	*cld_aeid*
AP_CLD_AEQ_BIT	Called AE Qualifier	*cld_aeq*
AP_CLD_APID_BIT	Called AP Invocation-identifier	*cld_apid*
AP_CLD_APT_BIT	Called AP Title	*cld_apt*
AP_REM_PADDR_BIT	Calling Presentation Address	*rem_paddr*
AP_PCDL_BIT	Presentation Context Definition List	*pcdl*
AP_PCDRL_BIT	Presentation Context Definition Result List	*pcdrl*
AP_DPCN_BIT	Default Presentation Context Name	*dpcn*
AP_QOS_BIT	Quality of Service	*qos*
AP_ACSE_SEL_BIT	ACSE Version Selector	*a_version_sel*

flag	parameter	field
AP_PRES_SEL_BIT	Presentation Version Selector	*p_version_sel*
AP_SESS_SEL_BIT	Session Version Selector	*s_version_sel*
AP_AFU_SEL_BIT	ACSE Requirements	*afu_sel*
AP_PFU_SEL_BIT	Presentation Requirements	*pfu_sel*
AP_SFU_SEL_BIT	Session Requirements	*sfu_sel*
AP_CLG_CONN_ID_BIT	Session-connection Identifier	*clg_conn_id*
AP_INIT_SYNC_PT_BIT	Initial Synchronization Serial Point Number	*init_sync_pt*
AP_INIT_TOKENS	Initial Token Assignment	*init_tokens*

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

If the ''X.410-1984'' mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the contents of *ubuf* buffer(s) is assumed to be a value of type SET encoded according to the Basic Encoding Rules for ASN.1. The SET value will have an implicit context-specific tag with a value of 1 (i.e., [1] IMPLICIT). Refer to the ISO Presentation Layer protocol definition (reference **ISO 8823**) for further information concerning the encoding of these values.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

A_ASSOC_RSP - used to respond to an association request indication

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the A_ASSOC_RSP primitive and restrictions on its use.

To send an A_ASSOC_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_A_ASSOC_RSP.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-information   */
                        /* field                        */
long res;               /* result of request            */
long diag;              /* reason (if rejected)         */
ap_a_assoc_env_t *env; /* environment attribute values */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

The argument, *cdata→res* must be one of the following:

AP_ACCEPT Accept the association.

AP_REJ_PERM Association permanently rejected.

AP_REJ_TRAN Association temporarily rejected.

The argument *cdata→diag* is used to indicate the reason for the result specified by *cdata→res*. If the result was AP_ACCEPT, this argument can be ignored. If the result was not AP_ACCEPT, the possible values for this argument are as follows:

AP_CLD_AEID_NREC Called AE invocation identifier not recognised.

AP_CLD_AEQ_NREC	Called AE qualifier not recognised.
AP_CLD_APID_NREC	Called AP invocation identifier not recognised.
AP_CLD_APT_NREC	Called AP Title not recognised.
AP_CLD_PADDR_UNK	Called presentation address unknown.
AP_CLG_AEID_NREC	Calling AE invocation identifier not recognised.
AP_CLG_AEQ_NREC	Calling AE qualifier not recognised.
AP_CLG_APID_NREC	Calling AP invocation identifier not recognised.
AP_CLG_APT_NREC	Calling AP Title not recognised.
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_DFCN_NSUP, AP_LCL_LIM_EXCEEDED, AP_TEMP_CONGESTION, or AP_UDATA_NREAD, the confirmation received by the remote user will indicate that the source of the association rejection was the Presentation service provider (see A_ASSOC_CNF). In this case, any Association-information passed with the A_ASSOC_RSP primitive is ignored.

If the ''X.410-1984'' mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→diag* argument is not used and any value specified by the user will be ignored.

The *cdata→env* argument can be used to override XAP environment attributes values used as parameters to the A-ASSOCIATE response service. If no attribute values are to be overidden, *cdata→env* may be set to *null*. Otherwise, *cdata→env* must point to an **ap_a_assoc_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask        */
ap_objid_t cntx_name;        /* AP_CNTX_NAME    */
ap_aei_api_id_t rsp_aeid;    /* AP_RSP_AEID     */
ap_aeq_t rsp_aeq;            /* AP_RSP_AEQ      */
ap_aei_api_id_t rsp_apid;    /* AP_RSP_APID     */
ap_apt_t rsp_apt;            /* AP_RSP_APT      */
ap_paddr_t rem_paddr;        /* AP_REM_PADDR    */
ap_cdrl_t pcdrl;             /* AP_PCDRL        */
long  dpcr;                  /* AP_DPCR         */
ap_qos_t qos;                /* AP_QOS          */
unsigned long afu_sel;       /* AP_AFU_SEL      */
unsigned long pfu_sel;       /* AP_PFU_SEL      */
unsigned long sfu_sel;       /* AP_SFU_SEL      */
ap_conn_id_t *cld_conn_id;   /* AP_CLD_CONN_ID  */
unsigned long init_sync_pt;  /* AP_INIT_SYNC_PT */
unsigned long init_tokens;   /* AP_INIT_TOKENS  */
```

The *mask* element of this structure is a bit mask indicating which environment attributes associated with this primitive are to be overridden. Values for this field are formed by OR'ing together zero or more of the flags listed in the table below. When a bit is set, the value of the associated parameter will be taken from the *cdata* argument rather than from the XAP environment. Otherwise, the value for the parameter contained in the XAP environment will be used. Specifying a value for a particular parameter in the *cdata* argument has the same effect on the value of the corresponding attribute in the XAP environment as calling *ap_set_env*( ). See the description of the related environment attributes for information on how the fields of this argument are used.

flag	parameter	field
AP_CNTX_NAME_BIT	Application Context Name	*cntx_name*
AP_RSP_AEID_BIT	Responding AE Invocation-identifier	*rsp_aeid*
AP_RSP_AEQ_BIT	Responding AE Qualifier	*rsp_aeq*
AP_RSP_APID_BIT	Responding AP Invocation-identifier	*rsp_apid*
AP_RSP_APT_BIT	Responding AE Title	*rsp_apt*
AP_REM_PADDR_BIT	Responding Presentation Address	*rem_paddr*
AP_PCDRL_BIT	Presentation Context Definition Result List	*pcdrl*
AP_DPCR_BIT	Default Presentation Context Result	*dpcr*
AP_QOS_BIT	Quality of Service	*qos*
AP_AFU_SEL_BIT	ACSE Requirements	*afu_sel*
AP_PFU_SEL_BIT	Presentation Requirements	*pfu_sel*
AP_SFU_SEL_BIT	Session Requirements	*sfu_sel*
AP_CLD_CONN_ID_BIT	Session-connection Identifier	*cld_conn_id*
AP_INIT_SYNC_PT_BIT	Initial Synchronization Serial Point Number	*init_sync_pt*
AP_INIT_TOKENS_BIT	Initial Token Assignment	*init_tokens*

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

If the X.410-1984 mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the contents of *ubuf* buffer(s) will be a value of type SET encoded according to the Basic Encoding Rules for ASN.1. If the association is being accepted, the SET value should have an implicit context-specific tag with a value of 1 (i.e. [1] IMPLICIT). If the association is being rejected, the SET value

should have been encoded with the universal SET tag (i.e. [UNIVERSAL 17]). Refer to the ISO Presentation Layer protocol definition (reference **ISO 8823**) for further information concerning the encoding of these values.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_DIAG] The value of *cdata→diag* is not valid.

[AP_BADCD_RES] The value of *cdata→res* is not valid.

NAME

A_ASSOC_CNF - used to confirm an association request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The A_ASSOC_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm the establishment of an association between two application entities.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the A_ASSOC_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_A_ASSOC_CNF.

cdata The following members of *cdata* are used for this primitive:

```
long res;                /* result of request      */
long res_src;            /* source of result       */
long diag;               /* reason (if rejected)   */
ap_a_assoc_env_t *env;   /* environment attributes */
```

cdata→res will be set to indicate the result of the association request. The possible values for *cdata→res* are as follows:

AP_ACCEPT	Accept the association.
AP_REJ_PERM	Association permanently rejected.
AP_REJ_TRAN	Association temporarily rejected.

The argument *cdata→res_src* indicates the source of the result and will be one of the following:

AP_ACSE_SERV_USER	ACSE service user.
AP_ACSE_SERV_PROV	ACSE service provider.
AP_PRES_SERV_PROV	Presentation service provider.
AP_SESS_SERV_PROV	Session service provider.
AP_TRAN_SERV_PROV	Transport service provider.

The argument *cdata→diag* is not meaningful 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 source of the result is AP_ACSE_SERV_USER, *cdata→diag* will be one of the following:

AP_CLD_AEID_NREC	Called AE invocation identifier not recognised.
AP_CLD_AEQ_NREC	Called QE qualifier not recognised.
AP_CLD_APID_NREC	Called AP invocation identifier not recognised.
AP_CLD_APT_NREC	Called AP Title not recognised.
AP_CLG_AEID_NREC	Calling AE invocation identifier not recognised.
AP_CLG_AEQ_NREC	Calling AE invocation identifier not recognised.
AP_CLG_APID_NREC	Calling AP invocation identifier not recognised.
AP_CLG_APT_NREC	Calling AP Title not recognised.
AP_CNTX_NAME_NSUP	Calling application context name not supported.
AP_NRSN	No reason given.
AP_NULL	Null.

If the source of the result is AP_ACSE_SERV_PROV, *cdata→diag* will be one of:

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

If the source of the result is AP_PRES_SERV_PROV, *cdata→diag* will be one of:

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

If the source of the result is AP_SESS_SERV_PROV, *cdata→diag* will be one of:

AP_CLD_SADDR_UNK	Called session address unknown.
AP_NRSN	Reason not specified.
AP_NULL	Null.
AP_SESS_PICS_ERROR	Implementation restriction stated in the PICS.
AP_SESS_VER_NSUP	Proposed protocol versions not supported.
AP_SPM_CONGESTION	SPM congestion at connect time.

AP_SSUSER_NATT	SS-user not attached to SSAP.
AP_TRAN_PROV	Could not establish transport connection.

Finally, if the source of the result is AP_TRAN_SERV_PROV, *cdata→diag* will be one of:

AP_CLD_TADDR_UNK	Address unknown.
AP_DUPSRCREF	Duplicate source reference for the same pair of NSAPs.
AP_HPLENINV	Header or parameter length invalid.
AP_MISMATCHREF	Mismatched references.
AP_NORMDISCON	Normal disconnect initiated by the TS-user.
AP_NRSN	Reason not specified.
AP_REFONNETCON	Connection refused on this network connection.
AP_REFOVERFLOW	reference overflow.
AP_TE_CONGESTION	Remote transport entity congestion at connect request time.
AP_TRAN_NEGFAIL	Connection negotiation failed (e.g., proposed classes not supported).
AP_TRAN_PERROR	Transport protocol error.
AP_TSAP_CONGESTION	Congestion at TSAP.
AP_TSUSER_NATT	Session entity not attached to TSAP.

If the ‘‘X.410-1984’’ mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→diag* argument will be set to AP_DIAG_NOVAL by the library.

Two conformant XAP Library implementations may return different source/diagnostic combinations under the same conditions. For example, an implementation might return either AP_PRES_SERV_PROV and AP_SESS_PROV, or AP_TRAN_SERV_PROV and AP_NRSN, if an association could not be established because of an unspecified problem that occurred while trying to set up the supporting transport connection. Moreover, because the use of reason codes is not well standardised, an application may receive different source/diagnostic combinations under the same conditions when attempting to establish an association to different OSI implementations through a single XAP Library implementation. Consequently, users are cautioned against placing dependencies on specific source/diagnostic combinations in their applications.

The *cdata→env* argument can be used to retrieve the values of the XAP environment attributes that correspond to parameters of A-ASSOCIATE confirmation service. returned in the *cdata* argument. If the AP_COPYENV attribute in the XAP environment is *false*, these values will not be returned in the *cdata* argument and *cdata→env* will be set to NULL when *ap_rcv*( ) returns. If AP_COPYENV is *true*, the XAP library will allocate an **ap_a_assoc_env_t** structure and any necessary substructures and return a pointer to it in *cdata→env*. The caller can release the storage allocated for the **ap_a_assoc_env_t** structure and its substructures by passing a pointer to *cdata* to *ap_free*( ). The following elements of

the **ap_a_assoc_env_t** structure are used for this primitive:

```
unsigned long mask;          /* bit mask        */
ap_objid_t cntx_name;        /* AP_CNTX_NAME    */
ap_aei_api_id_t rsp_aeid;    /* AP_RSP_AEID     */
ap_aeq_t rsp_aeq;            /* AP_RSP_AEQ      */
ap_aei_api_id_t rsp_apid;    /* AP_RSP_APID     */
ap_apt_t rsp_apt;            /* AP_RSP_APT      */
ap_paddr_t rem_paddr;        /* AP_REM_PADDR    */
ap_cdrl_t pcdrl;             /* AP_PCDRL        */
long  dpcr;                  /* AP_DPCR         */
ap_qos_t qos;                /* AP_QOS          */
unsigned long a_version_sel; /* AP_ACSE_SEL     */
unsigned long p_version_sel; /* AP_PRES_SEL     */
unsigned long s_version_sel; /* AP_SESS_SEL     */
unsigned long afu_sel;       /* AP_AFU_SEL      */
unsigned long pfu_sel;       /* AP_PFU_SEL      */
unsigned long sfu_sel;       /* AP_SFU_SEL      */
ap_conn_id_t *cld_conn_id;   /* AP_CLD_CONN_ID  */
unsigned long init_sync_pt;  /* AP_INIT_SYNC_PT */
unsigned long init_tokens;   /* AP_INIT_TOKENS  */
```

The *mask* element of this structure is a bit mask indicating which parameters associated with this primitive were received. Values for this field are formed by OR'ing together zero or more of the flags listed in the table below. When a bit is set, the specified parameter was received. Otherwise, the parameter was not received and the corresponding field in the **ap_a_assoc_env_t** structure is not set.

flag	parameter	field
AP_CNTX_NAME_BIT	Application Context Name	*cntx_name*
AP_RSP_AEID_BIT	Responding AE Invocation-identifier	*rsp_aeid*
AP_RSP_AEQ_BIT	Responding AE Qualifier	*rsp_aeq*
AP_RSP_APID_BIT	Responding AP Invocation-identifier	*rsp_apid*
AP_RSP_APT_BIT	Responding AE Title	*rsp_apt*
AP_REM_PADDR_BIT	Responding Presentation Address	*rem_paddr*
AP_PCDRL_BIT	Presentation Context Definition Result List	*pcdrl*
AP_DPCR_BIT	Default Presentation Context Result	*dpcr*
AP_QOS_BIT	Quality of Service	*qos*
AP_A_VERSION_SEL_BIT	ACSE Version Selector	*a_version_sel*
AP_P_VERSION_SEL_BIT	Presentation Version Selector	*p_version_sel*
AP_S_VERSION_SEL_BIT	Session Version Selector	*s_version_sel*
AP_AFU_SEL_BIT	ACSE Requirements	*afu_sel*
AP_PFU_SEL_BIT	Presentation Requirements	*pfu_sel*
AP_SFU_SEL_BIT	Session Requirements	*sfu_sel*
AP_CLD_CONN_ID_BIT	Session-connection Identifier	*cld_conn_id*
AP_INIT_SYNC_PT_BIT	Initial Synchronization Serial Point Number	*init_sync_pt*
AP_INIT_TOKENS_BIT	Initial Token Assignment	*init_tokens*

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

If the ''X.410-1984'' mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the contents of *ubuf* buffer(s) is assumed to be a value of

type SET encoded according to the Basic Encoding Rules for ASN.1. If the association was accepted, the SET value will have an implicit context-specific tag with a value of 1 (i.e., [1] IMPLICIT). If the association was rejected, the SET value is assumed to have been encoded with the universal SET tag (i.e., [UNIVERSAL 17]). Refer to the ISO Presentation Layer protocol definition (reference **ISO 8823**) for further information concerning the encoding of these values.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

A_PABORT_REQ - used to initiate a Presentation provider abort

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the A_PABORT_REQ primitive and restrictions on its use.

To send an A_PABORT_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_A_PABORT_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long rsn;    /* reason for abort        */
long evt;    /* event that caused abort */
```

Cdata→rsn must be set to indicate the reason for the abort. The possible values for *cdata→rsn* are as follows:

AP_INVAL_PPDU_PARM	Abort due to an invalid presentation protocol data unit parameter.
AP_NSPEC	The reason for the abort is not specified.
AP_RSN_NOVAL	No value was supplied for this optional parameter.
AP_UNEXPT_PPDU	Abort due to an unexpected presentation protocol data unit.
AP_UNEXPT_PPDU_PARM	Abort due to an unexpected presentation protocol data unit parameter.
AP_UNEXPT_SSPRIM	Abort due to an unexpected session service primitive.
AP_UNREC_PPDU	Abort due to an unrecognised presentation protocol data unit.
AP_UNREC_PPDU_PARM	Abort due to an unrecognised presentation protocol data unit parameter.

The *cdata→evt* field indicates which PPDU or Session primitive which triggered the abort. The possible values for *cdata→evt* are as follows:

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

If the ''X.410-1984'' mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→evt* argument is not used and any value specified by the user will be ignored.

ubuf Since no data is sent on an A_PABORT_REQ, this value must be *null*.

flags Not used.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_EVT] The value of *cdata→evt* is not valid.

[AP_BADCD_RSN] The value of *cdata→rsn* is not valid.

NAME

A_PABORT_IND - used to indicate a provider abort

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the A_PABORT_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_A_PABORT_IND.

cdata The following members of *cdata* are used for this primitive:

```
long rsn;    /* reason for abort        */
long evt;    /* event that caused abort */
long src;    /* source of abort         */
```

cdata→src indicates the source of the abort. Possible values for this field are as follows:

AP_PRES_SERV_PROV	Presentation service provider.
AP_SESS_SERV_PROV	Session service provider.
AP_TRAN_SERV_PROV	Transport service provider.

cdata→rsn will be set to indicate the reason for the abort. The possible values for *cdata→rsn* depend on the value of *cdata→src*. If *cdata→src* is set to AP_PRES_SERV_PROV, *cdata→rsn* will be set to one of the following:

AP_INVAL_PPDU_PARM	Abort due to an invalid presentation protocol data unit parameter.
AP_NSPEC	The reason for the abort is not specified.
AP_RSN_NOVAL	no value was supplied for this optional parameter.
AP_UNEXPT_PPDU	Abort due to an unexpected presentation protocol data unit.
AP_UNEXPT_PPDU_PARM	Abort due to an unexpected presentation protocol data unit parameter.

AP_UNEXPT_SSPRIM	Abort due to an unexpected session service primitive.
AP_UNREC_PPDU	Abort due to an unrecognised presentation protocol data unit.
AP_UNREC_PPDU_PARM	Abort due to an unrecognised presentation protocol data unit parameter.

If *cdata→src* is set to AP_SESS_SERV_PROV, *cdata→rsn* will be set to one of the following:

AP_SESS_PERROR	Abort due to session protocol error.
AP_SESS_PICS_ABORT	Abort due to a session implementation restriction stated in the PICS.
AP_SESS_TDISCON	Abort due to transport disconnect.
AP_SESS_UNDEFINED	Abort due to undefined session error.

If *cdata→src* is set to AP_TRAN_SERV_PROV, *cdata→rsn* will be set to one of the following:

AP_CLD_TADDR_UNK	Address unknown.
AP_DUPSRCREF	Duplicate source reference for the same pair of NSAPs.
AP_HPLENINV	Header or parameter length invalid.
AP_MISMATCHREF	Mismatched references.
AP_NORMDISCON	Normal disconnect initiated by the TS-user.
AP_NRSN	Reason not specified.
AP_REFONNETCON	Connection refused on this network connection.
AP_REFOVERFLOW	Reference overflow.
AP_TE_CONGESTION	Remote transport entity congestion at connect request time.
AP_TRAN_NEGFAIL	Connection negotiation failed (e.g., proposed classes not supported).
AP_TRAN_PERROR	Transport protocol error.
AP_TSAP_CONGESTION	Congestion at TSAP.
AP_TSUSER_NATT	Session entity not attached to TSAP.

If *cdata→src* is set to AP_PRES_SERV_PROV, the PPDU or Session primitive which triggered the abort will be indicated by *cdata→evt*. The possible values for *cdata→evt* are as follows:

AP_EVT_NOVAL	No value was supplied for this optional parameter.
AP_PEI_AC	Alter context PPDU.
AP_PEI_ACA	Alter context acknowledge PPDU.

AP_PEI_ARP	Abnormal release provider PPDU.
AP_PEI_ARU	Abnormal release user PPDU.
AP_PEI_CP	Connect presentation PPDU.
AP_PEI_CPA	Connect presentation accept PPDU.
AP_PEI_CPR	Connect presentation reject PPDU.
AP_PEI_RS	Resynchronize PPDU.
AP_PEI_RSA	Resynchronize acknowledge PPDU.
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_RELEASE_IND	Session release indication.
AP_PEI_S_RELEASE_CNF	Session release confirmation.
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_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.

If *cdata→src* is not set to AP_PRES_SERV_PROV, the *cdata→evt* field is not meaningful.

If the ''X.410-1984'' mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→evt* argument will be set to AP_EVT_NOVAL by the library.

ubuf No user data is associated with this primitive. The user data buffers pointed to by this argument are not updated.

flags Since no data is received with an A_PABORT_IND primitive, the AP_MORE bit of the **int** pointed to by *flags* will not be set when *ap_rcv*( ) returns.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

A_RELEASE_REQ - used to request the release an association

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the A_RELEASE_REQ primitive and restrictions on its use.

To send an A_RELEASE_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_A_RELEASE_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;  /* length of user-information field */
                    /* of APDU                          */
long rsn;           /* reason for release               */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→rsn, conveys the reason for the release request. The following values are allowed:

AP_REL_NORMAL	Normal release request.
AP_REL_URGENT	Urgent release request.
AP_REL_USER_DEF	A user defined release request.
AP_RSN_NOVAL	No value was specified for this optional parameter.

If the ‘‘X.410-1984’’ mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→rsn* argument is not used and any value specified by the user will be ignored.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_RSN] The value of *cdata→rsn* is not valid.

NAME

A_RELEASE_IND - used to indicate an association release request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the A_RELEASE_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ) the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_A_RELEASE_IND.

cdata The following members of *cdata* are used for this primitive:

```
long rsn;           /* reason for release */
```

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

Cdata→rsn, will indicate the reason for the release request. The possible values for *cdata→rsn* are given below.

AP_REL_NORMAL	Normal release request.
AP_REL_URGENT	Urgent release request.
AP_REL_USER_DEF	A user defined release request.
AP_RSN_NOVAL	No value was specified for this optional parameter.

Providing a reason for the release request is optional, if the release request did not contain a reason then *cdata→rsn* will be set to AP_RSN_NOVAL.

If the ‘‘X.410-1984’’ mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→rsn* argument will be set to AP_RSN_NOVAL by the library.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

A_RELEASE_RSP - used to respond to an association release request

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The A_RELEASE_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to an association release request.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the A_RELEASE_RSP primitive and restrictions on its use.

To send the A_RELEASE_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_A_RELEASE_RSP.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;  /* length of user-information  */
                    /* field of APDU               */
long res;           /* result                      */
long rsn;           /* reason for the result       */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

The argument, *cdata→res* is used to indicate whether the release request is accepted or rejected. Legal values for *cdata→res* are as follows:

AP_REL_AFFIRM Indicates acceptance of the release.

AP_REL_NEGATIVE 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* must be one of the following:

AP_REL_NORMAL Indicates a normal release.

AP_REL_NOTFINISHED Indicates the user is not finished with the association.

AP_REL_USER_DEF Indicates a user defined reason.

AP_RSN_NOVAL No value was supplied for this optional parameter.

If the ''X.410-1984'' mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→rsn* argument is not used and any value specified by the user will be ignored.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_RES] The value of *cdata→res* is not valid.

[AP_BADCD_RSN] The value of *cdata→rsn* is not valid.

NAME

A_RELEASE_CNF - used to confirm a release request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The A_RELEASE_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm the acceptance or rejection of a previously sent release request.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the A_RELEASE_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_A_RELEASE_CNF.

cdata The following members of *cdata* are used for this primitive:

```
long res;        /* result of the release request */
long rsn;        /* reason for the result         */
```

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

cdata→res will be set to indicate the result of the release request. The possible values for *cdata→res* are as follows:

AP_REL_AFFIRM	Indicates acceptance of the release request.
AP_REL_NEGATIVE	Indicates rejection of the release request. Note that this value will only be set if the Session negotiated release functional unit is selected.

The reason associated with the result will be indicated by *cdata→rsn*. The possible values for *cdata→rsn* are as follows:

AP_REL_NORMAL	Indicates a normal release.
AP_REL_NOTFINISHED	Indicates the remote user was not finished.
AP_REL_USER_DEF	Indicates a user defined reason.
AP_RSN_NOVAL	No value was supplied for this optional parameter.

Providing a reason for the response to the release request is optional. If the remote user chose not to supply a reason, *cdata→rsn* will be set to AP_RSN_NOVAL.

If the ''X.410-1984'' mode of operation is in effect (i.e., the AP_X410_MODE bit of AP_MODE_SEL is set), the *cdata→rsn* argument will be set to AP_RSN_NOVAL by the library.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_ACTDISCARD_REQ - request an activity be terminated with implication that activity's content is cancelled

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTDISCARD_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to provide the user with access to the S-ACTIVITY-DISCARD session service. The S-ACTIVITY-DISCARD session service allows the user to request the abnormal termination of the current activity. There is an implication that the work achieved before the activity was terminated is canceled; however this is not enforced by the Session provider.

Sending this primitive may result in the loss of undelivered data.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_ACTDISCARD_REQ primitive and restrictions on its use.

To send a P_ACTDISCARD_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd: This argument identifies the XAP Library instance being used.

sptype: This argument must be set to AP_P_ACTDISCARD_REQ.

cdata: The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
long rsn;               /* reason                    */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

The argument, *cdata→rsn*, is used to specify the reason for the activity discard. The following values are legal for this field:

AP_RCV_ABILITY_JEOP	SS-user receiving ability jeopardised (i.e., data received may not be handled properly).
AP_LCL_USER_ERR	Local SS-user error.

AP_SEQ_ERR	Sequence error.
AP_DEMAND_DT_TOK	Demand data token.
AP_UNRCVR_PROC_ERR	Unrecoverable procedure error.
AP_NSPEC_ERR	Non-specific error.

ubuf: Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

Note that when Session version 1 is in effect, no user-data may be sent with this primitive.

flags: The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_ACTDISCARD_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_RSN] The value of *rsn* is not valid.

NAME

P_ACTDISCARD_IND - indicate an activity was terminated with implication that activity's content is canceled

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTDISCARD_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the current activity was abnormally terminated. There is an implication that the work achieved before the activity was terminated is canceled; however this is not enforced by the Session provider.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_ACTDISCARD_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd: This argument identifies the XAP Library instance being used.

sptype: The **unsigned long** pointed to by this argument will be set to AP_P_ACTDISCARD_IND.

cdata: The following members of *cdata* are used for this primitive:

```
long rsn;           /* reason */
```

The argument, *cdata→rsn*, indicates the reason for the activity discard. The following values are legal for this field:

AP_RCV_ABILITY_JEOP	SS-user receiving ability jeopardised (i.e., data received may not be handled properly).
AP_LCL_USER_ERR	Local SS-user error.
AP_SEQ_ERR	Sequence error.
AP_DEMAND_DT_TOK	Demand data token.
AP_UNRCVR_PROC_ERR	Unrecoverable procedure error.
P_NSPEC_ERR	Non-specific error.

ubuf: Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags: The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_ACTDISCARD_RSP - used to respond to an activity discard indication

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTDISCARD_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond an activity discard request.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_ACTDISCARD_RSP primitive and restrictions on its use.

To send a P_ACTDISCARD_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd: This argument identifies the XAP Library instance being used.

sptype: This argument must be set to AP_P_ACTDISCARD_RSP.

cdata: The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
unsigned long tokens;   /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major / activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf: Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

Note that when Session version 1 is in effect, no user-data may be sent with this primitive.

flags: The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_ACTDISCARD_RSP primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_ACTDISCARD_CNF - confirms an activity discard request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTDISCARD_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm an activity discard request. Upon receipt of this primitive, all available tokens will be assigned to the user that initiated activity-discard service. AP_TOKENS_OWNED will be updated accordingly.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_ACTDISCARD_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd: This argument identifies the XAP Library instance being used.

sptype: The **unsigned long** pointed to by this argument will be set to AP_P_ACTDISCARD_CNF.

cdata: The following members of *cdata* are used for this primitive:

```
None
```

ubuf: Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags: The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_ACTEND_REQ - used to indicate the end of an activity

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTEND_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to provide the user with access to the S-ACTIVITY-END session service. The S-ACTIVITY-END session service allows the user to indicate the end of an activity.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_ACTEND_REQ primitive and restrictions on its use.

To send a P_ACTEND_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_ACTEND_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
long sync_p_sn;         /* serial number             */
                        /* (set by provider)         */
unsigned long tokens;   /* surrendered tokens        */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→sync_p_sn is the serial number assigned to this synchronization point. This field is set by the provider. When the XAP Library is being used asynchronously, *ap_snd*( ) may return before the value of the synchronization point serial number is received from the underlying session provider. In this case, the service returns the [AP_AGAIN] error code. The user must call *ap_snd*( ) repeatedly, with the same arguments, until the result SUCCESS is returned, at which point the *cdata→sync_p_sn* argument indicates the value assigned to the synchronization point by the session service provider.

cdata→tokens identifies the token(s) surrendered. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_ACTEND_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_ACTEND_IND - indicates the end of an activity

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTEND_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the end of an activity.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_ACTEND_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_ACTEND_IND.

cdata The following members of *cdata* are used for this primitive:

```
long sync_p_sn;     /* synchronization point */
                    /* serial number         */
```

The argument, *cdata→sync_p_sn*, is an integer between 0 and 999,998. The value of this argument is the serial number of the major synchronization point set as a result of ending the activity.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_ACTEND_RSP - used to respond to a request to end an activity

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTEND_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to a request to end an activity.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_ACTEND_RSP primitive and restrictions on its use.

To send a P_ACTEND_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_ACTEND_RSP.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;     /* length of user-data field */
unsigned long tokens;  /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_ACTEND_RSP primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_ACTEND_CNF - confirms a request to end an activity

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTEND_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm the end of an activity.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_ACTEND_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_ACTEND_CNF.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_ACTINTR_REQ - request an activity be terminated so that work can be resumed later

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTINTR_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to provide the user with access to the S-ACTIVITY-INTERRUPT session service. The S-ACTIVITY-INTERRUPT session service allows the user to request the abnormal termination of the current activity so that work achieved before the interruption is not cancelled and may be resumed later.

Sending this primitive may result in the loss of undelivered data.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_ACTINTR_REQ primitive and restrictions on its use.

To send a P_ACTINTR_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_ACTINTR_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
long rsn;               /* reason                    */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

The argument *cdata→rsn* is used to specify the reason for the activity interrupt. The following values are legal for this field:

AP_RCV_ABILITY_JEOP	SS-user receiving ability jeopardised (i.e., data received may not be handled properly).
AP_LCL_USER_ERR	Local SS-user error.
AP_SEQ_ERR	Sequence error.

AP_DEMAND_DT_TOK	Demand data token.
AP_UNRCVR_PROC_ERR	Unrecoverable procedure error.
AP_NSPEC_ERR	Non-specific error.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

Note that when Session version 1 is in effect, no user-data may be sent with this primitive.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_ACTINTR_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_RSN] The value of *rsn* is not valid.

NAME

P_ACTINTR_IND - indicate an activity was terminated so work can be resumed later

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTINTR_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the current activity was abnormally terminated, but that work achieved before the interruption was not canceled and may be resumed later.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_ACTINTR_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_ACTINTR_IND.

cdata The following members of *cdata* are used for this primitive:

```
long rsn;           /* reason      */
```

The argument, *cdata→rsn*, indicates the reason for the activity interrupt. The following values are legal for this field:

AP_RCV_ABILITY_JEOP	SS-user receiving ability jeopardised (i.e. data received may not be handled properly).
AP_LCL_USER_ERR	Local SS-user error.
AP_SEQ_ERR	Sequence error.
AP_DEMAND_DT_TOK	Demand data token.
AP_UNRCVR_PROC_ERR	Unrecoverable procedure error.
AP_NSPEC_ERR	Non-specific error.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_ACTINTR_RSP - used to respond to an activity interrupt indication

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTINTR_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond an activity interrupt request.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_ACTINTR_RSP primitive and restrictions on its use.

To send an P_ACTINTR_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_ACTINTR_RSP.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
unsigned long tokens;   /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

Note that when Session version 1 is in effect, no user-data may be sent with this primitive.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_ACTINTR_RSP primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_ACTINTR_CNF - confirms an activity interrupt request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTINTR_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm an activity interrupt request. Upon receipt of this primitive, all available tokens will be assigned to the user who initiated the activity-interrupt service. AP_TOKENS_OWNED will be updated accordingly.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_ACTINTR_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_ACTINTR_CNF.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_ACTRESUME_REQ - used to issue a request to resume an activity

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTRESUME_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to provide the user with access to the S-ACTIVITY-RESUME session service. The S-ACTIVITY-RESUME session service allows the user to indicate that a previously interrupted activity is resumed.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_ACTRESUME_REQ primitive and restrictions on its use.

To send an P_ACTRESUME_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_ACTRESUME_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;              /* length of user-data field */
ap_octet_string_t act_id;       /* activity identifier       */
ap_octet_string_t old_act_id;   /* old activity identifier   */
long sync_p_sn;                 /* synchronization point     */
                                /* serial number             */
ap_old_conn_id_t *old_conn_id;  /* old session connection    */
                                /* identifier                */
unsigned long tokens;           /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→act_id, is a **ap_octet_string_t** structure (defined in the *manual page* for Chapter 3 on page 33). *cdata→act_id.data* points to a buffer that contains up to 6 octets of information used as the new identifier for this activity. *cdata→act_id.length* specifies the length of the octet string (must be ≤ 6).

cdata→old_act_id is also an **ap_octet_string_t** structure and is subject to the same restrictions described above. This argument is the original identifier for the activity that is being resumed.

cdata→sync_p_sn must be an integer between 0 - 999,998.

Cdata→old_conn_id is a pointer to an **ap_old_conn_id_t** structure which is defined as

```
typedef struct {
  ap_octet_string_t *clg_user_ref; /* Calling SS-user      */
                                   /* Reference           */
  ap_octet_string_t *cld_user_ref; /* Called SS-user       */
                                   /* Reference           */
  ap_octet_string_t *comm_ref;     /* Common Reference     */
  ap_octet_string_t *addtl_ref;    /* Additional Reference */
} ap_old_conn_id_t;
```

This argument is used to identify the session connection in which the activity being resumed was originally started. The four members of this structure correspond to the calling SS-user reference, called SS-user reference, common reference and additional reference components of the old session connection identifier parameter of the S-ACTIVITY-RESUME request primitive. If no session connection identifier is to be sent, *cdata→old_conn_id* must be set to *null*.

Each of the members of the **ap_old_conn_id_t** structure are of type **ap_octet_string_t** (see above). The *clg_user_ref* and the *cld_user_ref* members 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 zero-length **ap_octet_string_t**.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_ACTRESUME_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_ACT_ID]	The value of *act_id* is not valid.
[AP_BADCD_OLD_ACT_ID]	The value of *old_act_id* is not valid.
[AP_BADCD_OLD_CONN_ID]	The value of *old_conn_id* is not valid.
[AP_BADCD_SYNC_P_SN]	The value of *sync_p_sn* is not valid.

NAME

P_ACTRESUME_IND - used to indicate a request to resume an activity

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTRESUME_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a request to resume an activity.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_ACTRESUME_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_ACTRESUME_IND.

cdata The following members of *cdata* are used for this primitive:

```
ap_octet_string_t  act_id;       /* activity identifier      */
ap_octet_string_t  old_act_id;   /* old activity identifier  */
long sync_p_sn;                  /* synchronization point    */
                                 /* serial number            */
ap_old_conn_id_t  *old_conn_id;  /* old session connection   */
                                 /* identifier               */
```

The argument, *cdata→act_id*, is a **ap_octet_string_t** structure (defined in the *manual page* for Chapter 3 on page 33). *cdata→act_id.data* points to a buffer that contains up to 6 octets of information used as the new identifier for this activity. *cdata→act_id.length* specifies the length of the octet string.

cdata→old_act_id is also an **ap_octet_string_t**. This argument is the original identifier for the activity that is being resumed.

cdata→sync_p_sn will be an integer between 0 - 999,998.

cdata→old_conn_id is pointer to an **ap_old_conn_id_t** structure (described in the *manual page* for *P_ACTRESUME_REQ* on page 166). This argument is used to identify the session connection in which the activity being resumed was originally started. If no value was received for this optional parameter, *cdata→old_conn_id* will be set to *null*.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_ACTSTART_REQ - used to issue a request to start a new activity

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTSTART_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to provide the user with access to the S-ACTIVITY-START session service. The S-ACTIVITY-START session service allows the user to indicate that a new activity is entered.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_ACTSTART_REQ primitive and restrictions on its use.

To send an P_ACTSTART_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_ACTSTART_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;        /* length of user-data field */
ap_octet_string_t act_id; /* activity identifier       */
unsigned long tokens;     /* surrendered tokens        */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

The argument, *cdata→act_id*, is a **ap_octet_string_t** structure (defined in the *manual page* for Chapter 3 on page 33). *cdata→act_id.data* points to a buffer that contains up to 6 octets of information used to identify this activity. *cdata→act_id.length* specifies the length of the octet string (must be ≤ 6).

cdata→tokens identifies the token(s) surrendered. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK Data token.

AP_MAJACT_TOK Synchronize-major/activity token.

AP_SYNCMINOR_TOK Synchronize-minor token.

AP_RELEASE_TOK Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_ACTSTART_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_ACT_ID] The value of *act_id* is not valid.

NAME

P_ACTSTART_IND - used to indicate a resynchronization request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_ACTSTART_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the beginning of a new activity.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_ACTSTART_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_ACTSTART_IND.

cdata The following members of *cdata* are used for this primitive:

```
ap_octet_string_t act_id;  /* activity identifier */
```

The argument, *cdata→act_id*, is a **ap_octet_string_t** structure (defined in the *manual page* for Chapter 3 on page 33). Upon return, *cdata→act_id.data* points to a buffer that contains up to 6 octets of information used to identify this activity. *Cdata→act_id.length* specifies the length of the octet string.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_CDATA_REQ - used to send capability data

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_CDATA_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to send capability data over an established association. The capability data transfer service enables data to be sent outside of an activity.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_CDATA_REQ primitive and restrictions on its use.

To send an P_CDATA_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_CDATA_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance. Where the underlying session connection has negotiated an unlimited TSDU size for the outgoing direction, the user need not set the *udata_length* field, as XAP can begin to format and send the appropriate session PDU without knowing how much user information is to follow.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_CDATA_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_CDATA_IND - used to indicate receipt of capability data

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_CDATA_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the receipt of capability data.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_CDATA_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_CDATA_IND.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_CDATA_RSP - used to send response to capability data

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_CDATA_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to a previously received capability data indication. The capability data transfer service enables data to be sent outside of an activity.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_CDATA_RSP primitive and restrictions on its use.

To send an P_CDATA_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_CDATA_RSP.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
unsigned long tokens;   /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance. Where the underlying session connection has negotiated an unlimited TSDU size for the outgoing direction, the user need not set the *udata_length* field, as XAP can begin to format and send the appropriate session PDU without knowing how much user information is to follow.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_CDATA_RSP primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_CDATA_CNF - used to confirm receipt of capability data

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_CDATA_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm receipt of capability data.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_CDATA_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_CDATA_CNF.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_CTRLGIVE_REQ - used to surrender the entire set of available tokens

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_CTRLGIVE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to provide the user with access to the S-CONTROL-GIVE session service. The S-CONTROL-GIVE session service allows the user to surrender the entire set of available tokens.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_CTRLGIVE_REQ primitive and restrictions on its use.

To send an P_CTRLGIVE_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_CTRLGIVE_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;    /* length of user-data field */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

After issuing this primitive, AP_TOKENS_OWNED will reflect the fact that all available tokens were surrendered to the remote user.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

Note that when Session version 1 is in effect, no user-data may be sent with this primitive.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_CTRLGIVE_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_CTRLGIVE_IND - indicates that the remote SS-user has surrendered all available tokens

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_CTRLGIVE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the remote SS-user has surrendered all available tokens.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_CTRLGIVE_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_CTRLGIVE_IND.

cdata The following members of *cdata* are used for this primitive:

```
None
```

Upon receipt of this primitive, AP_TOKENS_OWNED will be updated.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_DATA_REQ - used to send normal data

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_DATA_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to send user data over an established association.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_DATA_REQ primitive and restrictions on its use.

To send an P_DATA_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_DATA_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;     /* length of the user-information */
                       /* field                          */
unsigned long tokens; /* surrendered tokens              */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance. Where the underlying session connection has negotiated an unlimited TSDU size for the outgoing direction, the user need not set the *udata_length* field, as XAP can begin to format and send the appropriate session PDU without knowing how much user information is to follow.

cdata→tokens identifies the token(s) surrendered. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

Note that the P_DATA_REQ primitive may not be issued without one or more octets of user-data.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_TOKENS] The value of cdata→tokens is not valid.

[AP_NODATA] An attempt was made to send this primitive with no user-data.

NAME

P_DATA_IND - used to indicate receipt of normal data

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_DATA_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_DATA_IND.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_RESYNC_REQ - used to issue a resynchronise request

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_RESYNC_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request that the association be set to an agreed defined state.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_RESYNC_REQ primitive and restrictions on its use.

To send an P_RESYNC_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_RESYNC_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;              /* length of user-data    */
                                /* field                  */
long resync_type;               /* resynchronization type */
long sync_p_sn;                 /* synchronization point  */
                                /* serial number          */
unsigned long token_assignment; /* token assignment       */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

The argument *cdata→resync_type* indicates the type of resynchronization requested. The possible values for this argument are:

AP_ABANDON Resynchronization Type is *abandon*.

AP_RESTART Resynchronization Type is *restart*.

AP_SET Resynchronization Type is *set*.

The argument *cdata→sync_p_sn* depends upon the value of *cdata→resync_type*. If *cdata→resync_type* is set to AP_ABANDON, *cdata→sync_p_sn* is ignored. If *cdata→resync_type* is set to AP_RESTART, *cdata→sync_p_sn* must be set to a

synchronisation serial point number which is greater than or equal to the serial number of the last major synchronisation point and less than or equal to the value of the next synchronisation point number to be used. If *cdata→resync_type* is set to AP_SET, *cdata→sync_p_sn* may be any valid value from 0 - 999,999.

cdata→token_assignment is used to specify the desired token assignment after the resynchronisation. See Section 4.1.4 on page 60 for a discussion of how this parameter may be set. An invalid assignment for an available token causes *ap_snd*( ) to return an error.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_RESYNC_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_RESYNC_TYPE]	The value of *cdata→resync_type* is not valid.
[AP_BADCD_SYNC_P_SN]	The value of *cdata→sync_p_sn* is not valid.
[AP_BADCD_TOKENS]	The value of *cdata→tokens* is not valid.
[AP_NO_PRECEDENCE]	The resynchronisation requested by the local user does not have precedence over the one requested by the remote user.

NAME

P_RESYNC_IND - used to indicate a resynchronization request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p
```

DESCRIPTION

The P_RESYNC_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a request to set the association to an agreed defined state.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_RESYNC_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_RESYNC_IND.

cdata The following members of *cdata* are used for this primitive:

```
long resync_type;                /* resynchronization type */
long sync_p_sn;                  /* synchronization point  */
                                 /* serial number          */
unsigned long token_assignment; /* token assignment       */
```

cdata→resync_type will be set to indicate the type of resynchronization which is requested. The possible values for *cdata→resync_type* are as follows:

AP_ABANDON Resynchronization Type *abandon*.

AP_RESTART Resynchronization Type *restart*.

AP_SET Resynchronization Type *set*.

The argument *cdata→sync_p_sn* conveys the serial number of the synchronization point to which resynchronization is requested.

The argument *cdata→token_assignment* conveys the proposed token assignment following resynchronisation (see the *manual page* for P_RESYNC_REQ).

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_RESYNC_RSP - used to respond to a resynchronize request

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_RESYNC_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to a request that the association be set to an agreed defined state.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_RESYNC_RSP primitive and restrictions on its use.

To send a P_RESYNC_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_RESYNC_RSP.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;              /* length of user-data   */
                                /* field                 */
long sync_p_sn;                 /* synchronization point */
                                /* serial number         */
unsigned long tokens;           /* tokens requested      */
unsigned long token_assignment; /* token assignment      */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→sync_p_sn must be a valid synchronization point serial number in the range 0 - 999,999.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK Data token

AP_SYNCMINOR_TOK Synchronize-minor token

AP_MAJACT_TOK Synchronize-major/activity token

AP_RELEASE_TOK Release token.

cdata→token_assignment is used to specify the assignment of tokens which were identified as *acceptor's choice* on the P_RESYNC_IND primitive. See Section 4.1.4 on page 60 for a discussion of how this parameter may be set. An invalid assignment for an available *acceptor's choice* token causes *ap_snd*( ) to return an error.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_RESYNC_RSP primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_SYNC_P_SN] The value of *cdata→sync_p_sn* is not valid.

[AP_BADCD_TOKENS] The value of *cdata→tokens* is not valid.

NAME

P_RESYNC_CNF - used to confirm a resynchronization request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_RESYNC_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm a request to set the association to a defined state.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_RESYNC_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_RESYNC_CNF.

cdata The following members of *cdata* are used for this primitive:

```
long sync_p_sn;                  /* synchronization point */
                                 /* serial  number        */
unsigned long token_assignment;  /* token assignment      */
```

cdata→sync_p_sn will be a valid synchronization point serial number in the range 0 - 999,999.

cdata→token_assignment indicates the assignment of those tokens which were identified as *acceptor's choice* when the P_RESYNC_REQ primitive was sent (see the *manual page* for P_RESYNC_RSP).

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_SYNCMAJOR_REQ - used to request the setting of a major sync point

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_SYNCMAJOR_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request that a major synchronization point be set. Major synchronization points are used to structure the exchange of information as a series of dialogue units. A dialogue unit is a segment of communication which is logically separated from all communication before and after it. A major synchronization point indicates the end of one dialogue unit and the beginning of the next.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_SYNCMAJOR_REQ primitive and restrictions on its use.

To send an P_SYNCMAJOR_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_SYNCMAJOR_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;     /* length of user-data field       */
long sync_p_sn;        /* serial number (set by provider) */
unsigned long tokens; /* surrendered tokens               */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→sync_p_sn is the serial number assigned to this synchronization point. This field is set by the provider. When the XAP Library is being used asynchronously, *ap_snd*( ) may return before the value of the synchronization point serial number is received from the underlying session provider. In this case, the service returns the [AP_AGAIN] error code. The user must call *ap_snd*( ) repeatedly, with the same arguments, until the result SUCCESS is returned, at which point the *cdata→sync_p_sn* argument indicates the value assigned to the synchronization point by the session service provider.

cdata→tokens identifies the token(s) surrendered. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_SYNCMAJOR_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_SYNCMAJOR_IND - used to indicate a request to set a major sync point

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_SYNCMAJOR_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the remote service user has requested that a major synchronization point be set. Major synchronization points are used to structure the exchange of information as a series of dialogue units. A dialogue unit is a segment of communication which is logically separated from all communication before and after it. A major synchronization point indicates the end of one dialogue unit and the beginning of the next.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_SYNCMAJOR_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_SYNCMAJOR_IND.

cdata The following members of *cdata* are used for this primitive:

```
long sync_p_sn;   /* indicated sync. pt. serial no. */
```

cdata→sync_p_sn is the serial number of this synchronization point.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_SYNCMAJOR_RSP - used to respond to a major sync point request

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_SYNCMAJOR_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to a request to set a major synchronization point. Major synchronization points are used to structure the exchange of information as a series of dialogue units. A dialogue unit is a segment of communication which is logically separated from all communication before and after it. A major synchronization point indicates the end of one dialogue unit and the beginning of the next.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_SYNCMAJOR_RSP primitive and restrictions on its use.

To send an P_SYNCMAJOR_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_SYNCMAJOR_RSP.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;     /* length of user-data field */
unsigned long tokens;  /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_SYNCMAJOR_RSP primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_SYNCMAJOR_CNF - used to confirm a sync major request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_SYNCMAJOR_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm a request to set a major synchronization point. Major synchronization points are used to structure the exchange of information as a series of dialogue units. A dialogue unit is a segment of communication which is logically separated from all communication before and after it. A major synchronization point indicates the end of one dialogue unit and the beginning of the next.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_SYNCMAJOR_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_SYNCMAJOR_CNF.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_SYNCMINOR_REQ - used to request the setting of a minor sync point

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_SYNCMINOR_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request that a minor synchronization point be set. Minor synchronization points are used to structure the exchange of information within a dialogue unit. A dialogue unit is a segment of communication which is logically separated from all communication before and after it.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_SYNCMINOR_REQ primitive and restrictions on its use.

To send an P_SYNCMINOR_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_SYNCMINOR_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
long sync_type;         /* type of request           */
long sync_p_sn;         /* serial number             */
                        /* (set by provider)         */
unsigned long tokens;   /* surrendered tokens        */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→sync_type is a bit mask which determines whether confirmation is optional and whether data separation is required on this minor synchronisation point. Values for this field are formed by OR'ing together zero or more of the following flags:

AP_NO_CONFIRMATION No explicit confirmation is required.

AP_DATA_SEPARATION Data Separation is requested. Data separation may only be requested when the data separation functional unit has been selected.

cdata→sync_p_sn is the serial number assigned to this synchronization point. This field is set by the provider. When the XAP Library is being used asynchronously, *ap_snd* may return before the value of the synchronization point serial number is received from the underlying session provider. In this case the service returns the [AP_AGAIN] error code. The user must call *ap_snd*( ) repeatedly, with the same arguments, until the result SUCCESS is returned, at which point the *cdata→sync_p_sn* argument indicates the value assigned to the synchronization point by the session service provider.

cdata→tokens identifies the token(s) surrendered. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK Data token.

AP_MAJACT_TOK Synchronize-major/activity token.

AP_SYNCMINOR_TOK Synchronize-minor token.

AP_RELEASE_TOK Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_SYNCMINOR_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_SYNC_TYPE] The value of *cdata→sync_type* is not valid.

NAME

P_SYNCMINOR_IND - used to indicate a request to set a minor sync point

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_SYNCMINOR_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the remote service user has requested that a minor synchronization point be set. Minor synchronization points are used to structure the exchange of information within a dialogue unit. A dialogue unit is a segment of communication which is logically separated from all communication before and after it.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_SYNCMINOR_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_SYNCMINOR_IND.

cdata The following members of *cdata* are used for this primitive:

```
long sync_type;     /* type of request               */
long sync_p_sn;     /* indicated sync. pt. serial no. */
```

cdata→sync_type is a bit mask which indicates whether confirmation is optional and whether data separation is required on this minor sync point. Values for this field are formed by OR'ing together zero or more of the flags below. When a bit is set, the specified indication was received:

AP_NO_CONFIRMATION Explicit confirmation is not required.

AP_DATA_SEPARATION Data Separation is in operation.

Note: These are bit fields so none, one or both could be active; the absence of AP_NO_CONFIRMATION means that explicit confirmation is required.

Cdata→sync_p_sn is the serial number of this synchronization point.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_SYNCMINOR_RSP - used to respond to a minor sync point request

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_SYNCMINOR_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to a request to set a minor synchronization point. Minor synchronization points are used to structure the exchange of information within a dialogue unit. A dialogue unit is a segment of communication which is logically separated from all communication before and after it.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_SYNCMINOR_RSP primitive and restrictions on its use.

To send a P_SYNCMINOR_RSP primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_SYNCMINOR_RSP.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
long sync_p_sn;         /* confirmed sync. pt.       */
                        /* serial no.                */
unsigned long tokens;   /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→sync_p_sn must be set to the synchronization point serial number being confirmed. This number must be greater than or equal to the lowest unconfirmed synchronization point, and less than the next available synchronization point.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK Data token.

AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_SYNCMINOR_RSP primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_SYNC_P_SN] The value of *sync_p_sn* is not valid.

NAME

P_SYNCMINOR_CNF - used to confirm a sync minor request

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_SYNCMINOR_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm a request to set a minor synchronization point. Minor synchronization points are used to structure the exchange of information within a dialogue unit. A dialogue unit is a segment of communication which is logically separated from all communication before and after it.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_SYNCMINOR_CNF primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_SYNCMINOR_CNF.

cdata The following members of *cdata* are used for this primitive:

```
long sync_p_sn;      /* confirmed sync. pt. serial no. */
```

cdata→sync_p_sn will indicate the confirmed synchronization point serial number.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_TDATA_REQ - used to send typed data

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_TDATA_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to send typed data over an established association. Typed data transfers are subject to the same service restrictions as normal data transfers except they are not subject to token restrictions.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_TDATA_REQ primitive and restrictions on its use.

To send an P_TDATA_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_TDATA_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;  /* length of the user-information */
                    /* field                          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance. Where the underlying connection session has negotiated an unlimited TSDU size for the outgoing direction, the user need not set the udata_length field, as XAP can begin to format and send the appropriate session PDU without knowing how much user information is to follow.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

Note that the P_TDATA_REQ primitive may not be issued without one or more octets of user-data.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_NODATA] An attempt was made to send this primitive with no user-data.

NAME

P_TDATA_IND - used to indicate receipt of typed data

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_TDATA_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the receipt of typed data. Typed data transfers are subject to the same service restrictions as normal data transfers except they are not subject to token restrictions.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_TDATA_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_TDATA_IND.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_TOKENGIVE_REQ - used to give a Session token

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_TOKENGIVE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library 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.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_TOKENGIVE_REQ primitive and restrictions on its use.

To send an P_TOKENGIVE_REQ primitive, the arguments to *ap_snd*( ) must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_TOKENGIVE_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;     /* length of user-data field */
unsigned long tokens;  /* tokens surrendered        */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→tokens identifies the token(s) surrendered. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major / activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

Note that when Session version 1 is in effect, no user-data may be sent with this primitive.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_TOKENGIVE_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

CAVEAT

User data may not be sent on the P_TOKENGIVE_REQ primitive when Session version 1 is in effect.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_TOKENGIVE_IND - used to indicate receipt of newly acquired tokens

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_TOKENGIVE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library 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.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_TOKENGIVE_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_TOKENGIVE_IND.

cdata The following members of *cdata* are used for this primitive:

```
unsigned long tokens;   /* tokens acquired */
```

cdata→tokens indicates which tokens were received. The newly acquired tokens will be indicated by one or more of the following values OR'ed together:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_RELEASE_TOK	Release token.

In addition, the XAP Library environment attribute AP_TOKENS_OWNED will be updated to reflect this acquisition.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_TOKENPLEASE_REQ - used to request a Session token

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_TOKENPLEASE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library 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.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_TOKENPLEASE_REQ primitive and restrictions on its use.

To send an P_TOKENPLEASE_REQ primitive, the arguments to *ap_snd* must be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_TOKENPLEASE_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;     /* length of user-data field */
unsigned long tokens;  /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK	Data token.
AP_MAJACT_TOK	Synchronize-major / activity token.
AP_SYNCMINOR_TOK	Synchronize-minor token.

AP_RELEASE_TOK Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_TOKENPLEASE_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

Refer to the *manual page* for *ap_snd*( ) on page 93.

NAME

P_TOKENPLEASE_IND - used to indicate request for tokens

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

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

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_TOKENPLEASE_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_TOKENPLEASE_IND.

cdata The following members of *cdata* are used for this primitive:

```
unsigned long tokens;   /* tokens requested */
```

cdata→tokens, will indicate which token(s) was (were) requested by the remote service user. Each requested token is indicated by setting a different bit in the argument. Thus, *cdata→tokens* will be set to one or more of the following values OR'ed together:

AP_DATA_TOK	Data token.
AP_SYNCMINOR_TOK	Synchronize-minor token.
AP_MAJACT_TOK	Synchronize-major/activity token.
AP_RELEASE_TOK	Release token.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_XDATA_REQ - used to send expedited user data

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_XDATA_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to send from 1 to 14 octets of expedited data over an established association, in a single call to *ap_snd*( ) (with the AP_MORE flag unset). Expedited data is free from the token and flow control constraints that apply to normal, typed, and capability data transfers.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_XDATA_REQ primitive and restrictions on its use.

To send an P_XDATA_REQ primitive, the arguments to *ap_snd*( ) must be set as described below:

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_XDATA_REQ.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ), with the exception that setting the AP_MORE flag will result in the [AP_BADFLAGS] error being returned.

Note that the P_XDATA_REQ primitive may not be issued without one or more octets of user-data.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_NODATA] An attempt was made to send this primitive with no user-data.

NAME

P_XDATA_IND - used to indicate receipt of expedited data

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_XDATA_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the receipt of expedited user data. Expedited data is free from the token and flow control constraints that apply to normal, typed and capability data transfers.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_XDATA_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_XDATA_IND.

cdata The following members of *cdata* are used for this primitive:

```
None
```

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_PXREPORT_IND - indicate receipt of an exception report from the SS-provider

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_PXREPORT_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to provide access to the S-P-EXCEPTION-REPORT service of the session layer. The S-P-EXCEPTION-REPORT service permits users to be notified that a service cannot be completed due to SS-provider protocol errors or malfunctions.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_PXREPORT_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_PXREPORT_IND.

cdata The following members of *cdata* are used for this primitive:

```
long rsn;  /* reason    */
```

The argument *cdata→rsn* indicates the reason for the exception report. The following values are legal for this field:

AP_PROTOCOL_ERR Protocol error.

AP_NSPEC_ERR Non-specific error.

ubuf No user data is associated with this primitive. The user data buffers pointed to by this argument are not updated.

flags Since no data is received with an P_PXREPORT_IND primitive, the AP_MORE bit of the **int** pointed to by *flags* will not be set when *ap_rcv*( ) returns.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

NAME

P_UXREPORT_REQ - send an exception report to the remote user

SYNOPSIS

```
#include <xap.h>

int ap_snd (
    int fd,
    unsigned long sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t *ubuf,
    int flags,
    unsigned long *aperrno_p)
```

DESCRIPTION

The P_UXREPORT_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to provide the user with access to the S-U-EXCEPTION-REPORT session service. The S-U-EXCEPTION-REPORT session service allows the user to report an exception condition.

Refer to the table in the *manual page* description for *ap_snd*( ) on page 93 for information concerning the effects of sending the P_UXREPORT_REQ primitive and restrictions on its use.

To send an P_UXREPORT_REQ primitive, the arguments to *ap_snd* must be set as described below:

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to AP_P_UXREPORT_REQ.

cdata The following members of *cdata* are used for this primitive:

```
long udata_length;      /* length of user-data field */
long rsn;               /* reason                    */
unsigned long tokens;   /* requested tokens          */
```

Where this primitive is to be sent using a series of calls to *ap_snd*( ) with the AP_MORE flag set, *cdata→udata_length* should be set to the total number of octets of encoded user data that will be sent with this primitive. If the total number of octets of encoded user-data is not known this field may be set to –1. However, in some XAP implementations setting this field to –1 may significantly degrade performance as it requires the implementation to buffer data until a complete SPDU can be transmitted.

The total number of octets of encoded user-data that can be sent with this primitive may be subject to an implementation dependent restriction. Any such restriction will be stated in the CSQ for an implementation.

The argument *cdata→rsn* is used to specify the reason for the exception report. The following values are legal for this field:

AP_DEMAND_DT_TOK	Demand data token.
AP_LCL_USER_ERR	Local SS-user error.
AP_NSPEC_ERR	Non-specific error.
AP_RCV_ABILITY_JEOP	SS-user receiving ability jeopardised (i.e., data received may not be handled properly).
AP_SEQ_ERR	Sequence error.

AP_UNRCVR_PROC_ERR Unrecoverable procedure error.

cdata→tokens identifies the token(s) requested. Tokens are identified by OR'ing together one or more of the following values:

AP_DATA_TOK Data token.

AP_MAJACT_TOK Synchronize-major/activity token.

AP_SYNCMINOR_TOK Synchronize-minor token.

AP_RELEASE_TOK Release token.

ubuf Use of the *ubuf* argument is described in the *manual page* for *ap_snd*( ) on page 93.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_snd*( ) on page 93.

When issuing the P_UXREPORT_REQ primitive via multiple *ap_snd*( ) calls (using the AP_MORE bit feature), one or more octets of data must be sent on the final *ap_snd*( ) call of the sequence.

aperrno_p This must point to a location which will be set to an error code if a failure occurs.

RETURN VALUE

Refer to the *manual page* for *ap_snd*( ) on page 93.

ERRORS

In addition to those listed in the *manual page* for *ap_snd*( ) on page 93, the following error codes can be reported for this primitive:

[AP_BADCD_RSN] The value of the *rsn* field is invalid.

NAME

P_UXREPORT_IND - indicate receipt of an exception report from the remote user

SYNOPSIS

```
#include <xap.h>

int ap_rcv (
    int fd,
    unsigned long *sptype,
    ap_cdata_t *cdata,
    ap_osi_vbuf_t **ubuf,
    int *flags,
    unsigned long *aperrno_P)
```

DESCRIPTION

The P_UXREPORT_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the receipt of an exception report.

Refer to the table in the *manual page* description for *ap_rcv*( ) on page 80 for information concerning the effects of receiving the P_UXREPORT_IND primitive and restrictions on its use.

When issuing *ap_rcv*( ), the arguments must be set as described in the *manual page* for *ap_rcv*( ) on page 80. Upon return, the *ap_rcv*( ) arguments will be set as described below.

fd This argument identifies the XAP Library instance being used.

sptype The **unsigned long** pointed to by this argument will be set to AP_P_UXREPORT_IND.

cdata The following members of *cdata* are used for this primitive:

```
long rsn;           /* reason    */
```

The argument *cdata→rsn* indicates the reason for the exception report. The following values are legal for this field:

AP_DEMAND_DT_TOK	Demand data token.
AP_LCL_USER_ERR	Local SS-user error.
AP_NSPEC_ERR	Non-specific error.
AP_RCV_ABILITY_JEOP	SS-user receiving ability jeopardised (i.e., data received may not be handled properly).
AP_SEQ_ERR	Sequence error.
AP_UNRCVR_PROC_ERR	Unrecoverable procedure error.

ubuf Use of the *ubuf* parameter is described in the *manual page* for *ap_rcv*( ) on page 80.

flags The *flags* argument is used to control certain aspects of primitive processing as described in the *manual page* for *ap_rcv*( ) on page 80.

aperrno_p The location pointed to by the *aperrno_p* argument is set to an error code if a failure has occurred.

RETURN VALUE

Refer to the *manual page* for *ap_rcv*( ) on page 80.

ERRORS

Refer to the *manual page* for *ap_rcv*( ) on page 80.

Appendix A

XAP Header File

This appendix reproduces the basic structures of the **<xap.h>** header file.

```
/*
 *      xap.h
 */

/*
 *      Data structures for X/Open ACSE/Presentation
 *      Library environment attributes and cdata parameters.
 */

/*
 *      Environment data structures
 */

/*
 * The following ID numbers for each protocol are used to
 * distinguish #defines of various kinds for each layer,
 * such as primitive names, environment attribute names,
 * error codes, etc.
 */

#define AP_ASN1_ID (11)
#define AP_ID      (8)   /* Also used to indicate inclusion */
                         /* of XAP header file              */
#define AP_ACSE_ID (7)
#define AP_PRES_ID (6)
#define AP_SESS_ID (5)
#define AP_TRAN_ID (4)
#define AP_OS_ID   (0)

/*
 *      Object Identifier structure
 */

#define AP_MAXOBJBUF              12
#define AP_CK_OBJ_NULL(O)         ((O)->length ? 0 : 1)
#define AP_SET_OBJ_NULL(O)        ((O)->length = 0)
```

```
typedef struct {
    long length;
    union {
            unsigned char short_buf[AP_MAXOBJBUF];
            unsigned char *long_buf;
    } b;
} ap_objid_t;


/* AP_PCDL */
typedef struct {
    long  pci;
    ap_objid_t   *a_sytx;
    int  size_t_sytx;
    ap_objid_t  **m_t_sytx;
} ap_cdl_elt_t;

typedef struct {
    int   size;
    ap_cdl_elt_t  *m_ap_cdl;
} ap_cdl_t;



/* AP_DCN */
typedef struct {
    ap_objid_t      *a_sytx;
    ap_objid_t      *t_sytx;
} ap_dcn_t;



/* AP_PCDRL */
/* AP_PCDRL.res */
#define AP_ACCEPT                (0)
#define AP_USER_REJ              (1)
#define AP_PROV_REJ              (2)

/* AP_PCDRL.prov_rsn */
#define AP_RSN_NSPEC             (0)
#define AP_A_SYTX_NSUP           (1)
#define AP_PROP_T_SYTX_NSUP      (2)
#define AP_LCL_LMT_DCS_EXCEEDED  (3)
```

```
typedef struct {
    long  res;
    ap_objid_t   *t_sytx;
    long  prov_rsn;
} ap_cdrl_elt_t;

typedef struct {
    int size ;
    ap_cdrl_elt_t  *m_ap_cdl;
} ap_cdrl_t;


typedef struct {
    int size;
    unsigned char *udata;
} ap_aeq_t;


typedef struct {
    int size;
    unsigned char *udata;
} ap_apt_t;


typedef struct {
    int size;
    unsigned char *udata;
} ap_aei_api_id_t;


typedef struct  {
    long   length;
    unsigned char   * data;
} ap_octet_string_t;


#define AP_UNKNOWN    0
#define AP_CLNS       1
#define AP_CONS       2
#define AP_RFC1006    3


/* AP_BIND_PADDR, AP_LCL_PADDR, AP_REM_PADDR */
typedef struct {
    ap_octet_string_t nsap;  /* NSAPAddress                          */
    int nsap_type;           /* AP_UNKNOWN,  AP_CLNS,  AP_CONS,      */
                             /* AP_RFC1006, other = system dependent */
} ap_nsap_t;
```

```
typedef struct  {
    ap_octet_string_t       *p_selector;
    ap_octet_string_t       *s_selector;
    ap_octet_string_t       *t_selector;
    int    n_nsaps;
    ap_nsap_t               *nsaps;
} ap_paddr_t;



/* AP_DCS */
typedef struct {
    long  pci;
    ap_objid_t   *a_sytx;
    ap_objid_t   *t_sytx;
} ap_dcs_elt_t;

typedef struct {
    int size ;
    ap_dcs_elt_t  *dcs;
} ap_dcs_t;



/* AP_CLD_CONN_ID, AP_CLG_CONN_ID */
typedef struct {
    ap_octet_string_t *user_ref;   /* SS-user Ref.    */
    ap_octet_string_t *comm_ref;   /* Common Ref.     */
    ap_octet_string_t *addtl_ref;  /* Additional Ref. */
} ap_conn_id_t;



typedef struct {
    ap_octet_string_t *clg_user_ref; /* Calling SS-user Reference */
    ap_octet_string_t *cld_user_ref; /* Called SS-user Reference  */
    ap_octet_string_t *comm_ref;     /* Common Reference          */
    ap_octet_string_t *addtl_ref;    /* Additional Reference      */
} ap_old_conn_id_t;



/* AP_QOS */
#define AP_NO          0
#define AP_YES         1
#define AP_PRITOP      0
#define AP_PRIHIGH     1
#define AP_PRIMID      2
#define AP_PRILOW      3
#define AP_PRIDFLT     4
```

```
typedef struct {
    long           targetvalue;    /* target value              */
    long           minacceptvalue; /* limiting acceptable value */
} ap_rate_t;


typedef struct {
    ap_rate_t      called;         /* called rate  */
    ap_rate_t      calling;        /* calling rate */
} ap_reqvalue_t;


typedef struct {
    ap_reqvalue_t  maxthrpt;       /* maximum throughput        */
    ap_reqvalue_t  avgthrpt;       /* average throughput        */
} ap_thrpt_t;


typedef struct {
    ap_reqvalue_t  maxdel;         /* maximum transit delay     */
    ap_reqvalue_t  avgdel;         /* average transit delay     */
} ap_transdel_t;


typedef struct {
    ap_thrpt_t  throughput;      /* throughput                    */
    ap_transdel_t transdel;      /* transit delay                 */
    ap_rate_t   reserrorrate;    /* residual error rate           */
    ap_rate_t   transffailprob;  /* transfer failure probability  */
    ap_rate_t   estfailprob;     /* connection establ failure     */
                                 /* probability                   */
    ap_rate_t   relfailprob;     /* connection release failure    */
                                 /* probability                   */
    ap_rate_t   estdelay;        /* connection establishment delay */
    ap_rate_t   reldelay;        /* connection release delay      */
    ap_rate_t   connresil;       /* connection resilience         */
    unsigned int protection;     /* protection                    */
    int         priority;        /* priority                      */
    char        optimizedtrans;  /* optimized dialogue transfer   */
                                 /* value: AP_YES or AP_NO        */
    char        extcntl;         /* extended control              */
                                 /* value: AP_YES or AP_NO        */
} ap_qos_t;
```

```
/* AP_DIAGNOSTIC */

typedef struct {
    long rsn;       /* reason for the abort    */
    long evt;       /* event that caused abort */
    long src;       /* source of abort         */
    char *error;    /* textual message         */
} ap_diag_t;



/*
 *      ap_pollfd structure for 'fds' argument to ap_poll().
 */

typedef struct {
    int fd;         /* XAP instance identifier */
    short events;   /* requested events        */
    short revents;  /* returned events         */
} ap_pollfd_t;



/*
 * Vectored buffer definitions
 */

typedef struct {
     unsigned char *db_base;      /* beginning of buffer    */
     unsigned char *db_lim;       /* last octet+1 of buffer */
     unsigned char db_ref;        /* reference count        */
} ap_osi_dbuf_t ;


typedef struct ap_osi_vbuf ap_osi_vbuf_t;
struct ap_osi_vbuf {
    ap_osi_vbuf_t *b_cont;           /* next message block */
    unsigned char *b_rptr;           /* 1st octet of data  */
    unsigned char *b_wptr;           /* 1st free location  */
    ap_osi_dbuf_t *b_datap;          /* data block         */
} ;
```

```
/*
 * Cdata type definition
 */

typedef struct {
    long udata_length;              /* length of user-data field */
    long rsn;                       /* reason for activity or    */
                                    /* abort/release primitives  */
    long evt;                       /* event that caused abort   */
    long sync_p_sn;                 /* synchronization point     */
                                    /* serial number             */
    long sync_type;                 /* synchronization type      */
    long resync_type;               /* resynchronization type    */
    long src;                       /* source of abort           */
    long res;                       /* result of association or  */
                                    /* release request           */
    long res_src;                   /* source of result          */
    long diag;                      /* reason for association    */
                                    /* rejection                 */
    unsigned long tokens;           /* tokens identifier:        */
                                    /* 0 => "tokens absent"      */
    ap_a_assoc_env_t *env;          /* environment attribute     */
                                    /* values                    */
    ap_octet_string_t act_id;       /* activity identifier       */
    ap_octet_string_t old_act_id;   /* old activity identifier   */
    ap_old_conn_id_t *old_conn_id;  /* old session connection    */
                                    /* identifier                */
} ap_cdata_t;
```

```
typedef struct {
    unsigned long mask;             /* bit mask        */
    unsigned long mode_sel;         /* AP_MODE_SEL     */
    ap_objid_t cntx_name;           /* AP_CNTX_NAME    */
    ap_aei_api_id_t clg_aeid;       /* AP_CLG_AEID     */
    ap_aeq_t clg_aeq;               /* AP_CLG_AEQ      */
    ap_aei_api_id_t clg_apid;       /* AP_CLG_APID     */
    ap_apt_t clg_apt;               /* AP_CLG_APT      */
    ap_aei_api_id_t cld_aeid;       /* AP_CLD_AEID     */
    ap_aeq_t cld_aeq;               /* AP_CLD_AEQ      */
    ap_aei_api_id_t cld_apid;       /* AP_CLD_APID     */
    ap_apt_t cld_apt;               /* AP_CLD_APT      */
    ap_paddr_t rem_paddr;           /* AP_REM_PADDR    */
    ap_cdl_t pcdl;                  /* AP_PCDL         */
    ap_dcn_t dpcn;                  /* AP_DPCN         */
    ap_qos_t qos;                   /* AP_QOS          */
    unsigned long a_version_sel;    /* AP_ACSE_SEL     */
    unsigned long p_version_sel;    /* AP_PRES_SEL     */
    unsigned long s_version_sel;    /* AP_SESS_SEL     */
    unsigned long afu_sel;          /* AP_AFU_SEL      */
    unsigned long pfu_sel;          /* AP_PFU_SEL      */
    unsigned long sfu_sel;          /* AP_SFU_SEL      */
    ap_conn_id_t *clg_conn_id;      /* AP_CLG_CONN_ID  */
    ap_conn_id_t *cld_conn_id;      /* AP_CLD_CONN_ID  */
    unsigned long init_sync_pt;     /* AP_INIT_SYNC_PT */
    unsigned long init_tokens;      /* AP_INIT_TOKENS  */
    ap_aei_api_id_t rsp_aeid;       /* AP_RSP_AEID     */
    ap_aeq_t rsp_aeq;               /* AP_RSP_AEQ      */
    ap_aei_api_id_t rsp_apid;       /* AP_RSP_APID     */
    ap_apt_t rsp_apt;               /* AP_RSP_APT      */
    ap_cdrl_t pcdrl;                /* AP_PCDRL        */
    long  dpcr;                     /* AP_DPCR         */
} ap_a_assoc_env_t;



/*
 *     ap_val_t union for 'val' argument to ap_set_env()
 */

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

Glossary

abstract syntax
The abstract description of a set of data values. Used by an application-service-element to or application to define the data structures to be transferred. Usually 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.

AE
Application-entity.

AET
Application-entity-title.

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), whilst others provide services oriented to a particular application (e.g., CMISE - the common management information service element).

ASE
Application-service-element

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 basic 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 for transferring data and the *transfer syntaxes* to be 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 basic reference model. This layer provides services which allow AEs to organise and synchronise their interactions. In addition to the connection and data transfer services of the Transport Layer, the Session Layer provides orderly release, synchronisation, activity management and half-duplex operation.

Transport Layer
Fourth layer in the OSI basic 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