X/Open CAE Specification

Remote Operations Service Element (XAP-ROSE) API

X/Open Company Ltd.

X/Open CAE Specification

Remote Operations Service Element (XAP-ROSE) API

ISBN: 1-85912-060-1

X/Open Document Number: C408

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 open 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 so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, 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 to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These 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 advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk with the following in the Subject line:

```
request corrigenda; topic index
```

This will return the index of publications for which Corrigenda exist.

This Document

This document is an X/Open CAE Specification. It specifies the Remote Operation Service Element (ROSE) access method for OSI user applications using the ISO ROSE services defined in ISO 9072-1 and ISO 9072-2. Since ISO 13712 is defined to be compatible with ISO 9072-1 and ISO 9072-2, this API also provides access to the services defined in ISO 13712.

The exposed library interface, XAP-ROSE, is composed of a set of ROSE primitives and attributes designed to be used along with the XAP Service Library. This (ROSE) subset of Service primitives and attributes is based on the objectives in the XAP-ACSE/Presentation Service Library, and the standards in the ISO IS 9072 ROSE Definition & Protocol publications.

XAP-ROSE is not meant as an alternative to XAP, but rather as a superset of available services, for those applications requiring access to the Remote Operations Service ASE. The main benefit from these extensions is the elimination of redundant code within applications using the ISO ROSE.

The ROSE service user referencing this document is presumed to be familiar with the X/Open ACSE/Presentation Services (XAP) API (see reference **XAP**).

Structure

- **Chapter 1** gives an introduction to XAP-ROSE. It includes an overview of the ROSE service, its relationship to the XAP Specification (see reference **XAP**), and compliance requirements for implemetations claiming conformance to this XAP-ROSE Specification. These requirements include the minimum set of functions that must be provided by an implementation of XAP-ROSE, as well as the requirements placed on the underlying ROSE, ACSE, & Presentation protocol providers.
- **Chapter 2** provides a description of XAP-ROSE, covering the ROSE model and environment, and an outline of its usage
- **Chapter 3** presents information on the XAP-ROSE environment
- **Chapter 4** presents the manual page descriptions for the XAP-ROSE functions
- **Chapter 5** presents the manual page descriptions for the XAP-ROSE primitives.

Intended Audience

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

API Implementors
: System vendors who implement an OSI 7 layer protocol stack providing in addition the ISO ROSE may use this specification to develop an XAP-ROSE compliant library interface to their protocol providers. This provides a portability path for support of applications from third party software vendors.

Applications Implementors
: Independent Software Vendors, (ISVs), which implement OSI and Non-OSI applications which require the use of an Invoke / Response paradigm, and which are to run in an ISO OSI (network) compliant environment. Taking advantage of a standard interface to the ROSE API enhances the portability of the application(s) across the different implementations of the ISO OSI protocol stack from multiple system vendors.

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.

Trade Marks

X/Open® is a registered trade mark, and the ‘‘X’’ device is a trade mark, of the X/Open Company Ltd.

UNIX® is a registered trade mark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Referenced Documents

The following documents are referenced in this specification:

ISO 8326
ISO 8326: 1987, Information Processing Systems — Open Systems Interconnection — Basic Connection-oriented Session Service Definition.

ISO 8327
ISO 8327: 1987, Information Processing Systems — Open Systems Interconnection — Basic Connection-oriented Session Protocol Specification.

ISO 8649
ISO 8649: 1988, Information Processing Systems — Open Systems Interconnection — Service Definition for the Association Control Service Element.

ISO 8650
ISO 8650: 1992 Information Processing Systems — Open Systems Interconnection — Protocol Specification for the Association Control Service Element.

ISO 8822
ISO 8822: 1988, Information Processing Systems — Open Systems Interconnection — Connection-oriented Presentation Service Definition.

ISO 8823
ISO 8823: 1988, Information Processing Systems — Open Systems Interconnection — Connection-oriented Presentation Protocol Specification.

ISO / IEC 9072
ISO / IEC 9072: 1989, Information Processing Systems — Text Communication — Remote Operations — Parts 1 and 2:

Part 1: Model, Notation and Service Definition
Part 2: Protocol Specification.

ISO 13712
ISO 13712, Remote Operations Service Element Specification.

XAP
X / Open CAE Specification, September 1993, ACSE / Presentation Services API (XAP) (ISBN: 1-872630-91-X, C303).

Chapter 1

Introduction

This Specification defines the X/Open ROSE programming interface (XAP-ROSE), which is an interface to the services provided by the ISO 9072-1 and ISO 9072-2 Remote Operations Service Elements. These reside in the Application layer of the ISO Open Systems Interconnect Reference Model.

1.1 Objectives

X/Open has defined an API to ACSE/Presentation Services, (XAP), which provides a programming interface to the ISO ACSE and Presentation services defined in the ISO 7-layer Open Systems Interconnect Reference Model. The XAP specification is independent of the implementation of the underlying protocol engines, providing a clear path for ISO application portability.

In order to provide a consistent method of access to the ISO 9072-1 and ISO 9072-2 (and ISO 13712) Remote Operations Service (ROSE), X/Open has defined a set of extensions to the XAP interface which allow applications to exploit the facilities provided by ROSE.

The primary benefit of these extensions is the reduction of code needed to provide each application with ROSE facilities. A well defined interface to ROSE - (a ROSE API) - makes available to Independent Software Vendors (ISVs) consistency of target protocol providers which is an incentive for further application development, that is, X.500 DSP/DAP, 1988 X.400 MS & RUA.

Another motivation is to provide a base (ROSE) API for the development of higher level procedures. These higher level procedures could be viewed as libraries of objects which are designed to conceal the complexities of the underlying communications architecture. There is already activity in the commercial (user) community which mandates this very requirement, (easier/intuitive access to OSI services).

An issue considered when defining this API addresses where a ROSE API should reside, and it's visibility to the application developer. The XAP extensions are not intended to be a high level interface. The XAP interface itself is not viewed as an end user interface, but simply as a well defined entry point to the OSI Protocol Stack, (in this case, offering the ROSE Protocol access), intended to be used by development staff that are very familiar with the requirements of the underlying Protocol providers.

The XAP-ROSE extensions are a straightforward method of defining the use of ROSE services by those applications which require the use of an request/response paradigm.

1.2 Relationship to the XAP Specification

This specification defines a set of extensions to the established X/Open ACSE/Presentation (XAP) API - see reference **XAP**. It is intended to be read in conjunction with the XAP Specification, and so does not reiterate information that is already presented in the XAP Specification.

The reader of this specification is presumed to be familiar with the first three chapters of the XAP Specification, as well as the following ISO publications:

ISO 9072-1 Remote Operations Service Definition

ISO 9072-2 Remote Operations Protocol Specification

ISO 8649 Association Control Service Definition

ISO 8650 Association Control Protocol Specification

ISO 8822 Presentation Service Definition

ISO 8823 Presentation Protocol Specification

ISO 8326 Session Service Definition

ISO 8327 Session Protocol Specification

ISO 13712 Remote Operations Service Element Specification

Functions and Primitives which are unchanged from the XAP definition are not repeated in this specification, and the reader is referred to XAP for clarification when necessary.

1.3 Scope and Limitations

XAP-ROSE provides an API to the services of the Remote Operations Service Element, as defined by ISO 9072-1 and ISO 9072-2. In addition, as ISO 13712 is defined to be compatible with ISO 9072-1 and ISO 9072-2, the API also provides access to the services defined in ISO 13712.

The ROSE service provider to which XAP-ROSE provides access is layered directly on top of the ACSE and Presentation Service.

The P-DATA service provides an unconfirmed transfer service.

No support is provided for an implementation of ROSE services using the RTSE ASE to provide a confirmed transfer service.

1.4 Overview of the ROSE Service

The XAP-ROSE API provides access to the services of the Remote Operations Service Element (ROSE), as defined in the referenced (ISO 9072-1 and ISO 9072-2) ROSE standard. The remainder of this specification assumes that the reader is familiar with the terms and concepts defined in the ROSE standard.

The XAP specification, upon which XAP-ROSE depends, provides an overview of the services to which that API provides access. This section provides a similar brief overview of the concepts associated with OSI Remote Operations.

The purpose of the ROSE standard is to support interactive applications in a distributed open systems environment. To do this, the standards define a Remote Operation notation, the ROSE application-service-element and the ROSE protocol.

1.4.1 Remote Operation Model

The Remote Operations Model describes a remote operation as an operation requested by one entity (the *invoker*), which another entity (the *performer*) attempts to perform and then reports the outcome of the attempt.

The model elaborates on this by introducing the concept of *linked-operations*, where the definition of an operation may include a list of *child-operations* that the performer of the *parent-operation* may call in order to accomplish the requested task. The performer of the child-operation is the invoker of the parent operation. This sequence may be recursive, as required.

An operation is classified according to whether it is expected to report its outcome and, if it is, according to whether operation invocation/reply is synchronous or asynchronous. This results in five classes of operation. However, classes 3-5, which do not return results in all cases, are only likely to be useful to a restricted group of specialised applications.

Operation Class 1: synchronous, reporting success or failure

Operation Class 2: asynchronous, reporting success or failure

Operation Class 3: asynchronous, reporting failure only (if any)

Operation Class 4: asynchronous, reporting success only (if any)

Operation Class 5: asynchronous, outcome not reported.

In the OSI Basic Reference Model, interaction between a pair of application processes is achieved via *application-entities*. The general model of an application-entity consists of a *user-element*, which uses a set of one or more *application-service-elements* (ASEs) to achieve the communications functions of the application process. This communication occurs in the context of an *application-association* that is created for the purposes of communication between the two application-entities.

In the Remote Operations model, these user-elements communicate using *operations* which are invoked via a conceptual *operation-interface*. A set of operations represents an application-specific ASE. Such an ASE is implemented by application-specific code using the services of the Remote Operations Service Element (ROSE) for invoking and replying to operations, and ACSE or RTSE for the management of application-associations.

This model is illustrated in Figure 1-1, which is based upon a similar figure in the referenced ROSE standard.

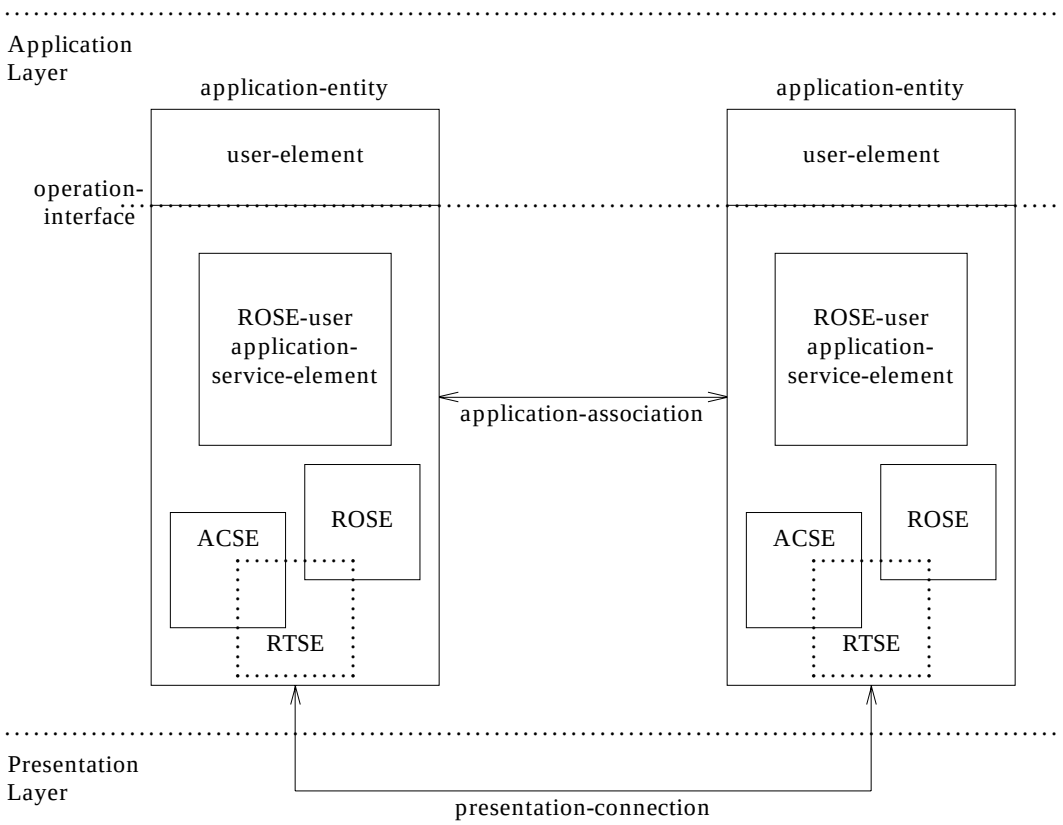

Figure 1-1 Remote Operations Model

The model classifies application-associations according to which of the application-entities are allowed to invoke operations:

Association Class 1: only the association initiator.

Association Class 2: only the association responder.

Association Class 3: both application-entities.

To support linked operations, a class 3 association is required.

1.4.2 Remote Operation Notation

The referenced ROSE standards define a Remote Operation notation (RO-notation) which is analagous to the interface definition language or notation that forms part of many remote procedure call mechanisms. It allows an application-service-element (ASE) to be defined for the support of a particular application or group of applications. This ASE consists of:

- a *bind* operation, which establishes an application-association between two application-entities for the invocation of operations
- one or more operations that may be invoked to implement an interactive protocol between the parties of the association
- an *unbind* operation, which releases the application-association.

As part of the assocation setup, ACSE is used to establish an application-context that identifies the set of ROSE-user ASEs and other ASEs that are to be available to the application-entities involved in the association.

As an example, the Directory Services defines a ROSE-user application-service-element using the RO-notation to provide a set of services such as *Read* and *Modify-Entry* for interogating or modifying the contents of a directory.

The RO-notation makes use of the data syntax notation and macro notation of Abstract Syntax Notation One (ASN.1). The macros provided allow the definition of the data syntax of an optional argument that may be passed to an operation, and an optional result that it may return when the operation is successful. They also allow the definition of the error codes that an unsuccessful operation may return.

1.4.3 ROSE Application Service Element

The purpose of an application-service-element is to provide services that will be of use to a number of applications. In the case of the ROSE ASE, the services support the implementation of Remote Operations.

An application-specific function uses the services provided by ROSE to implement the operations that are made available to the user element. In addition, the *bind* and *unbind* operations are implemented using the services provided by ACSE - or by Reliable Transfer Service (RTSE) if that ASE is being used to provide the transfer service for remote operations.

This application-specific function is analagous to the stub routines that are generated by some remote procedure call mechanisms, for use by an application when the interface definition is compiled.

ROSE provides unconfirmed services to invoke an operation, to return a reply for a successful invocation, and to return an error code for a failed operation. In addition, ROSE provides user and provider reject services to handle unexpected error conditions and protocol violations.

The ROSE service primitives are analagous to the remote procedure call functions that are used to transport call invocations and replies in some remote procedure call mechanisms.

1.4.4 ROSE Protocol

The ROSE protocol defines the PDUs that are exchanged to implement the services of the ROSE ASE, and the use of underlying services to transfer these PDUs. The ROSE uses a *transfer service* which may be the unconfirmed P-DATA primitive of the Presentation Layer, or the equivalent services of the Reliable Transfer Service ASE (RTSE).

In the case where ROSE protocol is implemented over RTSE, additional parameters passed in the ROSE service primitives are used by the ROSE protocol machine to manage the use of the RTSE services.

Unlike most application-service-elements, the ROSE protocol does not define a distinct abstract syntax for the encoding of its PDUs. Instead, it provides a set of abstract syntax definitions that are imported into the abstract syntax(es) that define the set of operations used by an application. ROSE PDUs are then encoded, along with any argument or result data associated with the operation being invoked or replied to, using the transfer syntax that the application has negotiated for use with the abstract syntax.

1.5 Terminology

Definition of Terms

The terminology used in this specification is that of the ISO standards which define the services to which XAP-ROSE provides access. For convenience, the abbreviations have been defined in the Glossary for this specification.

Use of Naming Prefixes

In order to preserve uniqueness, all functions, primitives, typedefs, data items and constants defined by this specification have names that have the format ap_ro_xxx (or AP_RO_XXX). The items that have the format ap_xxx (or AP_XXX) are defined in the XAP specification.

Alignment with ISO C

As part of aligning X/Open specifications with ISO C, this specification uses the ISO C function declaration syntax.

1.6 XAP-ROSE Compliance

All XAP-ROSE functions listed in Section 4.1.1 on page 25 must be supported.

All XAP-ROSE Primitives listed in Table 2-1 on page 18 must be supported.

An implementation which complies with this Specification must also comply with the conformance clauses of the ISO protocol specifications which this specification references:

- ISO 8822, 8823
  ISO Presentation Service and Protocol definitions
- ISO 8649, 8650
  ISO ACSE Service and Protocol definitions
- ISO 9072-1 and ISO 9072-2
  ISO ROSE Service and Protocol definitions.

These clauses specify requirements on combinations of functional units and, by implication, permitted sequences of primitives.

Conformance to the underlying protocols is stated in the appropriate Protocol Implementation Conformance Statement (PICS). PICS Proforma for the ISO/IEC Presentation, ACSE and ROSE 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.

International Standardized Profile ISO/IEC ISP 11183 specifies requirements on Presentation, ACSE and ROSE Protocols when they are used for OSI Management:

- ISP 11183-1: OSI Management - Specifications of ACSE, Presentation and Session Protocols for the use by ROSE and CMISE
- ISP 11183-2: OSI Management - Basic Management
- ISP 11183-3: OSI Management - Enhanced Management Communications.

A series of International Standardized Profiles is in preparation to specify Common Upper Layer Requirements. When they are approved by ISO/IEC:

- ISP 11188-1 will specify ‘‘Basic connection oriented requirements’’
- ISP 11188-2 will specify ‘‘Basic connection oriented requirements with ROSE’’
- ISP 11188-3 will specify ‘‘Minimal Upper Layer Facilities’’.

These ISPs are intended to be used in conjunction with specific Application Profiles (for example A-Profiles for the ISO Directory).

Support for specific profiles shall be declared by the vendor in the Conformance Statement Questionnaire (CSQ).

An implementation that conforms to the requirements of this specification must also comply with the mandatory requirements stated in the referenced XAP specification.

1.7 Future Directions

This XAP-ROSE CAE specification supports ROSE service providers that are implemented using the P-DATA service of the presentation layer directly to provide an unconfirmed transfer service for ROSE PDUs.

A future version of this specification may be produced which supports ROSE service providers that are implemented using the services of the RTSE ASE to provide a confirmed transfer service.

Support for RTSE-based service providers is expected to require the following facilities:

- access to the RTSE services RT-OPEN and RT-CLOSE
- use of the extended **ap_ro_cdata_t** parameters *priority* and *op_class*, which are defined by this specification but currently not used.

Support of RTSE-based services is not expected to affect the portability of existing applications, because no changes are expected to be required to the interfaces defined by this specification.

Chapter 2

Description of XAP-ROSE

This chapter describes the ROSE extensions to XAP, XAP-ROSE. It discusses usage of its functions, and use of the primitives associated with the XAP-ROSE interface. A brief description of how the interface may be used to establish an application association, perform remote operations, and release the association, is provided.

2.1 XAP-ROSE Model

Figure 2-1 shows how the interface presented by this API relates to the services of the OSI upper layers, and to the other X/Open-defined APIs to OSI services.

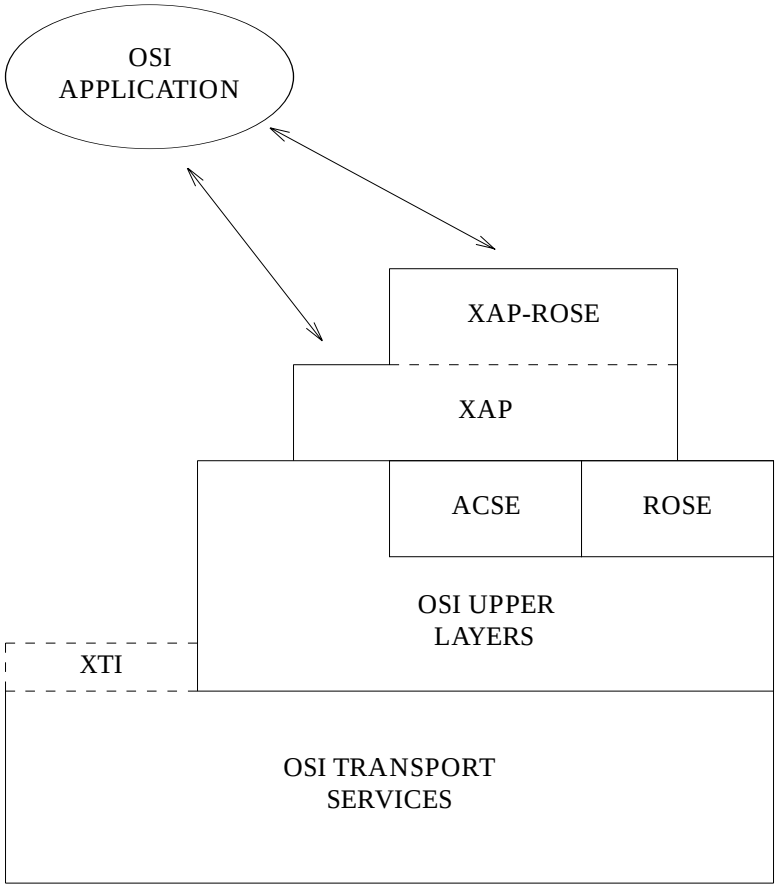

Figure 2-1 OSI Service Interfaces

The XAP-ROSE API is an extension of the XAP API. The model upon which that API is based also applies to the XAP-ROSE API. The following subsections describe the relationship between XAP and XAP-ROSE by showing how the XAP model applies to XAP-ROSE, and how XAP-ROSE extends that model to provide access to the services of the ROSE service.

This section of the specification should be read in conjunction with the equivalent section of the XAP Specification.

2.1.1 Service User

In XAP, an application program that uses the XAP API corresponds to the OSI term *application-entity*. The application code which uses the API may correspond to the OSI concept of a user-element or another application-service-element which accesses the services of the ASCE service and the Presentation Layer by using the XAP API.

XAP-ROSE extends the model of the XAP Service User to support the Remote Operations (RO) model defined in the referenced Remote Operations standards. In the RO model, the user of the ACSE and ROSE services is an application-service-element that implements an application-specific mapping of a set of remote operations onto the ACSE and ROSE services. These operations are made available to a user-element via a conceptual operation interface.

Thus, in the XAP-ROSE model, the *service-user* is the application code that implements the mapping of remote operations to ROSE services.

Note: XAP-ROSE does not constrain the structure of the application that uses the interface. For example, the remote operations may implement the conceptual operation interface as a set of stub functions that may be bound into an application using a remote procedure call mechanism. Alternatively, the remote operations may form an integral part of an application such as an implementation of a Directory Services Directory User Agent.

2.1.2 Service Provider

In XAP-ROSE, the service provider provides access to the services of the ROSE service plus those of the ASCE service and the Presentation Layer.

The *sevice-provider-identifier*, specified by the service-user when the instance is created, is used to select a particular ROSE service provider.

Currently, XAP-ROSE only supports ROSE implementations that are layered directly on top of the ACSE/Presentation Layers. See Section 1.7 on page 8 for information regarding possible support of the ROSE service layered on top of an RTSE-based transfer service.

2.1.3 XAP-ROSE Instance

An XAP-ROSE instance encompasses all of the capabilities of an XAP instance, plus additional capabilities required to support the services of the ROSE service using an existing application-association.

2.1.4 XAP-ROSE Environment

The XAP-ROSE environment contains all the environment attributes defined by the XAP environment, plus additional attributes that are required to support and control the operation of the ROSE service provider.

2.1.5 Service Primitive Parameters

XAP-ROSE defines additional elements of control data that are required to support and control the operation of the ROSE service provider.

2.1.6 User Data

The user data passed to or returned from XAP-ROSE service primitives represents the argument, result or error-parameter service primitive parameters. The actual data values passed by a particular remote operation are defined using the RO-notation defined in the referenced ROSE standards.

As in XAP, these data values must be encoded and decoded by the service user. However, the rules for encoding differ from those defined for the XAP primitives.

2.2 Establishing and Releasing an XAP-ROSE Instance

An XAP instance is established using the XAP function, *ap_open*( ). The *service provider* argument to *ap_open*( ) may be used to identify a service provider that supports ROSE. The XAP environment variable AP_MODE_AVAIL for such an association has the AP_ROSE_MODE flag set to indicate that this instance is capable of operating as an XAP-ROSE instance.

To use the instance as an XAP-ROSE instance, it is necessary to set the AP_ROSE_MODE flag in the XAP environment variable AP_MODE_SEL, at which point the functions, environment attributes and primitives defined by this specification become available to the application. When the AP_ROSE_MODE flag in AP_MODE_SEL is set, the environment attributes defined by XAP-ROSE are initialised to their default state.

Clearing the AP_ROSE_MODE flag in AP_MODE_SEL causes the XAP-ROSE functions, environment attributes and primitives to become unavailable. Alternatively, the XAP-ROSE instance can be destroyed using the XAP *ap_close*( ) function. Any state held by the ROSE service provider for an instance is lost when the flag is cleared. If the flag is set again for the instance at a later time, the ROSE service provider is set to its initial state and environment variables are re-initialised to their default values.

Currently, XAP-ROSE only supports service providers that implement the ROSE service using the services of the presentation layer directly as an unconfirmed transfer service. A future version of this API may support Reliable Transfer (RTSE) service providers.

2.3 Specifying ROSE Abstract Syntaxes

In order for the ROSE service provider to detect incoming ROSE PDUs and indicate them to the service user, the ROPM must examine each incoming P_DATA_IND primitive to determine if it contains a ROSE PDU.

As the abstract syntaxes that may contain ROSE PDUs are defined by the application as part of the specification of remote operations, the service user must inform the service provider which abstract syntaxes may contain ROSE PDUs.

To support this feature, XAP-ROSE defines an additional environment attribute AP_RO_PCI_LIST, that allows the service user to register the list of abstract syntaxes to be examined. The attribute consists of a list of object identifiers that are to be examined by the ROSE service provider. This attribute may be set calling the XAP function *ap_set_env*( ):

```
ap_set_env( fd, AP_RO_PCI_LIST, val, aperrno_p ) ;
```

where *val* is a structure of type **ap_val_t**, and *val.v* contains a pointer to a structure of type **ap_ro_pci_list_t**.

2.4 Controlling the ROSE Protocol Machine

Selecting AP_ROSE_MODE in the XAP environment attribute AP_MODE_SEL for an XAP instance makes the XAP-ROSE functions, environment attributes and primitives available to the service user. However, to enable the ROSE Procotol Machine (ROPM) and permit ROSE primitives to be sent and received, the service user must call the *ap_ro_init*( ) function. The user may call the *ap_ro_release*( ) function to disable the ROPM.

For XAP-ROSE instances used only for supporting the transfer of ROSE Operations, the ROPM may be enabled when the XAP-ROSE instance is created, and left so for the lifetime of the instance.

The ROPM is disabled automatically when the XAP-ROSE instance is destroyed by calling *ap_close*( ), so it is not necessary to call *ap_ro_release*( ) for this mode of usage.

An XAP-ROSE instance may be used to support a sequence of associations. It is not necessary to enable and disable the ROPM for each association. Once enabled, it becomes active whenever an established association exists, and is dormant at other times.

It is possible to use a single XAP instance and/or association to transfer both ROSE PDUs and those of other application protocols. Because examining incoming primitives for the presence of ROSE PDUs implies at least some processing overhead, an application which uses an XAP instance in this manner may use *ap_ro_init*( ) and *ap_ro_release*( ) to enable the ROPM only when required.

A call to *ap_ro_init*( ) causes the list of ROSE abstract syntaxes identified by the PCI entries in the environment attribute AP_RO_PCI_LIST to be validated.

The service user should ensure that the set of PCIs available on the association has been defined before this call (either in AP_DCS if the association is already established, or in AP_PCDL/AP_PCDRL in a state where the defined context set is not available). If this is not the case, the call to *ap_ro_init*( ) returns the error [AP_RO_CNTX_NOT_PRES] and the ROPM will not be enabled.

There are three different scenarios when *ap_ro_init*( ) can be called, and each operates as follows:

- For an initiator before connection establishment, the user sets AP_PCDL to contain the list of proposed presentation contexts, each PCI having an abstract syntax and one or more transfer syntaxes. The user sets AP_RO_PCI_LIST to contain the list of PCIs that identify presentation contexts using the ROSE protocol. The user then calls *ap_ro_init*( ) and the following checks are performed:

  — Each PCI in the AP_RO_PCI_LIST is checked against the AP_PCDL. If the PCI is not in the AP_PCDL an [AP_RO_BAD_PCI] error is returned. If the PCI is in the AP_PCDL, then each transfer syntax is checked to see if it is supported. If not, the transfer syntax is removed from the AP_PCDL. If no more tranfer syntaxes remain in the presentation context, the PCI is removed from the AP_PCDL. If there are no more PCIs in the AP_PCDL, an [AP_RO_CNTX_NOT_PRES] error is returned.

  Upon successful completion, the ROPM is enabled. If an error is returned, the ROPM is not enabled and the user may examine/modify the resulting AP_PCDL value before retrying.

- For an initiator or responder after connection establishment, the user sets AP_RO_PCI_LIST to contain the list of PCIs which identify presentation contexts using the ROSE protocol. The user then calls *ap_ro_init*( ) and the following checks are performed:

  — Each PCI in the AP_RO_PCI_LIST is checked against the PCIs in the AP_DCS. If the PCI is not in the AP_DCS an [AP_RO_BAD_PCI] error is returned. If the PCI is in the

AP_DCS, then the transfer syntax in the AP_DCS entry for this PCI is checked to see if it is supported. If it is not supported, an [AP_RO_T_SYTX_NSUP] error is returned.

Upon successful completion, the ROPM is enabled. If an error is returned, the ROPM is not enabled and the user may examine/modify the AP_RO_PCI_LIST value before retrying.

- For a responder during inward connection establishment, the user sets AP_PCDRL to accept or reject the proposed presentation contexts and tranfer syntaxes. The user then sets AP_RO_PCI_LIST to contain the list of PCIs that identify presentation contexts using the ROSE protocol. The user then calls *ap_ro_init*( ) and the following checks are performed:

  — Each PCI in the AP_RO_PCI_LIST is checked against the AP_PCDL. If the PCI is not in the AP_PCDL an [AP_RO_BAD_PCI] error is returned. If the PCI is in the AP_PCDL and the context for this PCI is accepted in the AP_PCDRL, the transfer syntax is checked. If the transfer syntax is not supported by the ROSE provider, the context is marked as rejected in the AP_PCDRL. If no contexts with supported transfer syntaxes remain, an [AP_RO_CNTX_NOT_PRES] error is returned.

  Upon successful completion, the ROPM is enabled. If an error is returned, the ROPM is not enabled and the user may examine/modify the AP_RO_PCI_LIST value before retrying.

The *ap_ro_init*( ) function may be called when ROPM is already enabled, in order to validate and install a modified list of ROSE abstract syntaxes.

XAP does not support the P-ALTER-CONTEXT service. If it is necessary to change the value of AP_RO_PCI_LIST during the lifetime of an association, the list of abstract syntaxes negotiated when the assocation was created must be sufficient to support the new value of AP_RO_PCI_LIST.

2.5 Implementing ROSE Bind and Unbind Operations

The ROSE service definition defines a *bind* operation that creates an association over which subsequent operation requests and replies are transferred. The ROSE service definition also defines an *unbind* operation that destroys the association created by the *bind* operation.

Depending on the version of the ROSE specification implemented by the XAP-ROSE provider, the *bind* and *unbind* operations may either be defined as a macros using the A-ASSOCIATE and A-RELEASE primitives, with the ROSE protocol encoded in the user information portion of these primitives, or as the separate ROSE primitives BIND and UNBIND.

In the XAP-ROSE API, if the underlying ROSE implementation supports the BIND and UNBIND primitives, the AP_RO_BIND_XXX and AP_RO_UNBIND_XXX primitives can be used as described in the following sections. Their usage is similar to that of other ROSE primitives defined through XAP-ROSE.

In the case where the A-ASSOCIATE and A-RELEASE primitives are to be used, the XAP-ROSE API user can implement the *bind/unbind* operation using the ACSE primitives defined by XAP. The rules for encoding the operation arguments for these primitives are those defined by XAP for the relevant primitives, rather than the rules defined in this specification for encoding other operation and reply arguments.

2.5.1 Bind and Unbind Encoding

Section 3.2.1 on page 23 describes how the user can determine availability of the ROSE BIND / UNBIND primitives in the XAP-ROSE provider.

The ROSE service specification ISO 9071-2 has macro definitions for the *bind* and *unbind* operations. These macros show the required tagging of the user information portion of the ACSE A-ASSOCIATE and A-RELEASE primitives when embedded in the Association-information EXTERNAL encoding. For each of the *bind* and *unbind* macros, an example encoding is shown below. These encodings take precedence over the encoding shown on the *ap_snd*( ) manual page of the XAP specification.

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

- bind macro for the A_ASSOC_REQ primitive:

```
user-information [30] {
  [EXTERNAL] { -- Association-information
    direct-reference [OBJECT IDENTIFIER] { ... }
    single-ASN1-type [0] { Argument-value [16] ... }
} }
```

- bind-accept macro for the A_ASSOC_RSP(accept) primitive:

```
user-information [30] {
  [EXTERNAL] { -- Association-information
    indirect-reference [INTEGER] { ... }
    single-ASN1-type [0] { Result-value [17] ... }
} }
```

- bind-refuse macro for the A_ASSOC_RSP(refuse) primitive:

```
user-information [30] {
  [EXTERNAL] { -- Association-information
    direct-reference [OBJECT IDENTIFIER] { ... }
    single-ASN1-type [0] { Error-value [18] ... }
} }
```

- unbind macro for the A_RELEASE_REQ primitive:

```
user-information [30] {
  [EXTERNAL] { -- Association-information
    indirect-reference [INTEGER] { ... }
    single-ASN1-type [0] { Argument-value [19] ... }
} }
```

- unbind-accept macro for the A_RELEASE_RSP(accept) primitive:

```
user-information [30] {
  [EXTERNAL] { -- Association-information
    indirect-reference [INTEGER] { ... }
    single-ASN1-type [0] { Result-value [20] ... }
} }
```

- unbind-refuse macro for the A_RELEASE_RSP(refuse) primitive:

```
user-information [30] {
  [EXTERNAL] { -- Association-information
    indirect-reference [INTEGER] { ... }
    single-ASN1-type [0] { Error-value [21] ... }
} }
```

The ASN.1 Basic Encoding Rules (BER) must be used to encode the user-information with the exception of the contents of the single-ASN1-type which must be encoded in the transfer syntax identified by direct-reference or indirect-reference. The presence of the direct-reference or indirect-reference is normally mandated by Application Profile.

2.6 XAP-ROSE Events/Primitives

The following list of Events make up the Valid set of XAP-ROSE events, which drive the XAP-ROSE and the associated ROPM (Remote Operations Protocol Machine).

XAP-ROSE Event	Send	Receive
AP_RO_BIND_REQ	Send	-
AP_RO_BIND_IND	-	Receive
AP_RO_BIND_RSP	Send	-
AP_RO_BIND_CNF	-	Receive
AP_RO_INVOKE_REQ	Send	-
AP_RO_INVOKE_IND	-	Receive
AP_RO_RESULT_REQ	Send	-
AP_RO_RESULT_IND	-	Receive
AP_RO_ERROR_REQ	Send	-
AP_RO_ERROR_IND	-	Receive
AP_RO_REJECTU_REQ	Send	-
AP_RO_REJECTU_IND	-	Receive
AP_RO_REJECTP_IND	-	Receive
AP_RO_UNBIND_REQ	Send	-
AP_RO_UNBIND_IND	-	Receive
AP_RO_UNBIND_RSP	Send	-
AP_RO_UNBIND_CNF	-	Receive

Table 2-1 XAP-ROSE Events / Primitives

2.6.1 Sending Primitives

An XAP-ROSE instance can be used to send *_REQ and *_RSP primitives defined both by this specification and by the XAP specification (subject to limitations imposed by the underlying ACSE or Presentation Layer service provider).

The XAP function *ap_snd*( ) is used to send all outbound primitives. The cdata argument for the *ap_snd*( ) call points to a *void *cdata* instead of *ap_cdata_t *cdata*. This allows the use of XAP as well as XAP-ROSE primitives to be combined in one Library. The *cdata* must point to an **ap_ro_cdata_t** structure if any ROSE-specific information is to be included with the primitive being sent. For specific **ap_ro_cdata_t** usage, refer to the primitive manual pages in Chapter 5.

When calling *ap_snd*( ) to send a ROSE primitive, the service user must indicate the abstract syntax that defines the data values contained in the PDU (both the ROSE protocol control data and any element which is specific to the remote operation, such as operation-argument). To do this, the service user sets the *cdata.pci* element to identify one of the abstract syntaxes which was defined as a ROSE abstract syntax using the AP_RO_PCI_LIST environment attribute.

The service provider performs the following steps to send a ROSE primitive:

- encode the ROSE-defined parts of the PDU (using the transfer syntax negotiated for the indicated presentation context)
- if required, append the data value that the user supplied in the ubuf parameter (this is specific to the remote operation, such as operation-argument)

- embed the resulting data value into a presentation data value (PDV) labelled with the relevant presentation context identifier
- pass the resulting PDV to the P_DATA request primitive as the User Data parameter.

Note: A data value passed to the service provider in the *ubuf* paramater is passed in encoded form. Unlike XAP primitives, no ACSE or Presentation PDU tag information is encoded with the data value; the buffers passed contain only the encoding of the data value itself. The reason for the deviation is that the ROSE service has to do the encoding of the full PDU.

2.6.2 Receiving Primitives

An XAP-ROSE instance can be used to receive the *_IND and *_CNF primitives defined both by this specification and by the XAP specification (subject to limitations imposed by the underlying ACSE or Presentation Layer service provider).

The XAP function *ap_rcv*( ) is used to receive all incoming primitives. The cdata argument for the *ap_rcv*( ) call points to a *void *cdata* instead of *ap_cdata_t *cdata*. This allows the use of XAP as well as XAP-ROSE primitives to be combined in one Library. The *cdata* must point to an **ap_ro_cdata_t** structure. For specific **ap_ro_cdata_t** usage, refer to the primitive manual pages in Chapter 5.

When receiving primitives, the ROSE service provider filters incoming primitives, using the list of abstact syntaxes defined by the environment attribute AP_RO_PCI_LIST, to detect incoming primitives that contain ROSE PDUs:

- a TD PPDU (Transfer Data Presentation PDU) that contains a single data value from one of the abstract syntaxes defined by the AP_RO_PCI_LIST environment attribute (where that data value is one of the defined ROSE PDU data types) is processed by the ROSE service provider and delivered to the service user as the appropriate ROSE primitive indication
- all other ACSE or Presentation PDUs are delivered to the service user as the appropriate ACSE or Presentation Layer indication or confirmation primitive.

Where the incoming TD PPDU does contain a ROSE PDU, the ROSE service provider performs the following steps to deliver the primitive to the service user:

- extract the presentation context identifier from the received presentation data value (PDV) and use it to decode the ROSE-defined PDU parameters for return in the *cdata* parameter
- set the *cdata.pci* element to indicate the presentation context to be used by the service user to decode any data value that is specific to the remote operation (such as operation-argument)
- set the *udata* parameter to return any data value that is specific to the remote operation (the encoded data value is returned)
- return the relevant ROSE primitive indication.

Note: A data value returned by the service provider in the *ubuf* paramater is passed in encoded form. Unlike XAP primitives, no ACSE or Presentation PDU tag information is encoded with the data value, the buffers passed contain only the encoding of the data value itself. The reason for the deviation is that the ROSE service has to do the encoding of the full PDU.

2.6.3 Encoding User Data for ROSE Primitives

The ROSE protocol specification ISO 9072-2 has ASN.1 definitions for the encodings of the ROSE operations. It is the responsibility of the XAP-ROSE user to encode/decode the following structure members for the following primitives:

primitive	member
AP_RO_INVOKE_REQ	operation-value, argument
AP_RO_RESULT_REQ	operation-value, result
AP_RO_ERROR_REQ	error-value, parameter

The XAP-ROSE provider is responsible for the encoding/decoding of the remainder of all primitives.

In the case of the *operation-value* and *error-value* members, which are of the ASN.1 type **OBJECT IDENTIFIER**, when passed in the global element of *cdata→value*, the contents should be encoded using ASN.1 BER.

During negotiation of Transfer Syntaxes, the XAP-ROSE provider only accepts Transfer Syntaxes supported by the implementation. However, from a practical perspective, ASN.1 BER should be supported to allow interoperability between the two systems. For the *Invoke*, *Result* and *Error* operations, the user must provide data encoded using the negotiated transfer syntax selected for the given Presentation Context Identifier (PCI) when the ROSE primitive associated with the operation is sent, and must decode the user data portion when a ROSE primitive is received.

The following extract from ISO 9072-1 shows the operation ASN.1 structures and shows in bold print the portion for which the XAP-ROSE user is responsible.

- AP_RO_INVOKE_REQ operation encoding

```
[SEQUENCE] {
  invoke-ID [INTEGER] { ... }
  linked-ID [INTEGER] { ... }
  operation-value [OBJECT IDENTIFIER] { ... }
  -- operation-value global encoding choice
  argument { ... } -- ANY DEFINED BY operation-value
  -- argument is provided by the user
}
```

- AP_RO_RESULT_REQ operation encoding

```
[SEQUENCE] {
  invoke-ID [INTEGER] { ... }
  [SEQUENCE] {
    operation-value [OBJECT IDENTIFIER] { ... }
    -- operation-value global encoding choice
    result { ... } -- ANY DEFINED BY operation-value
    -- result is provided by the user
  } -- OPTIONAL
}
```

- AP_RO_ERROR_REQ operation encoding

```
[SEQUENCE] {
  invoke-ID [INTEGER] { ... }
  error-value [OBJECT IDENTIFIER] { ... }
  -- error-value global encoding choice
  parameter { ... } -- ANY DEFINED BY error-value
  -- parameter is provided by the user
}
```

2.7 Using the XAP-ROSE Interface

The following is a brief summary of the steps required to establish an association with a remote ROSE user (application entity) using XAP-ROSE. This summary is only intended as a general description of how the XAP-ROSE might be used, and is not intended to be viewed as the paradigm to be followed by all ROSE user applications.

2.7.1 Obtain an XAP-ROSE Instance

An XAP-ROSE instance must be obtained / initialised by issuing an *ap_open*( ).

2.7.2 Initialise the XAP-ROSE Environment

The environment of the created instance is intialised by using either the *ap_init_env*( ) or *ap_restore*( ) XAP functions. The set of abstract syntaxes to be examined for incoming ROSE PDUs is defined by setting the AP_RO_PCI_LIST enviroment attribute, using the XAP function *ap_set_env*( ).

The ROSE service provider is activated by calling the *ap_ro_init*( ) function, at which point the service provider will begin examining received P_DATA_IND primitives to identify incoming ROSE PDUs.

2.7.3 Bind the XAP-ROSE user to a local PSAP address

The XAP-ROSE user must bind to a local PSAP address before attempting to issue any ROSE or ACSE service primitives. See Section 2.5 on page 16.

2.7.4 Send or Receive AP_RO_BIND_XXX or A_ASSOC_XXX Event

After the XAP-ROSE user has initialised all relevant environment attributes, it is ready to establish an application association. This may be done by either issuing an AP_RO_BIND_REQ / A_ASSOC_REQ, or by receiving and processing an AP_RO_BIND_IND / A_ASSOC_IND.

2.7.5 Transferring ROSE Primitives

Once the association has been established, the Application Entities may begin transferring information, either via the XAP-ROSE primitives, or by using the available XAP primitives, that is:

P_DATA_REQ, P_SYNCMINOR_REQ, etc...

2.7.6 Releasing the Application Association

Upon completion of information transfer, the applications may release the association via the AP_RO_UNBIND_REQ / A_RELEASE_REQ or the AP_RO_UNBIND_IND / A_RELEASE_IND Services.

Chapter 3

Environment

This chapter presents the additional environment attributes that XAP-ROSE defines in order to support and control a ROSE service provider.

3.1 Modified XAP Environment Attributes

XAP-ROSE modifies the syntax and semantics of the following XAP environment attributes:

AP_MODE_AVAIL: an additional flag AP_ROSE_MODE is defined for this environment attribute. If set, the service provider specified in the call to *ap_open*( ) is capable of providing the ROSE service. If this flag is set, the equivalent flag in AP_MODE_SEL may be set to use the XAP-ROSE features of the service provider.

AP_MODE_SEL: an additional flag AP_ROSE_MODE is defined for this environment attribute. Setting this flag causes the functions, environment attributes and primitives defined by this specification are initialised to their default state and become available to the service user.

Clearing the AP_ROSE_MODE flag in AP_MODE_SEL causes the XAP-ROSE functions, environment attributes and primitives to become unavailable, and, if enabled, disables the ROSE protocol machine (ROPM).

3.2 Environment Attributes Specific to XAP-ROSE

3.2.1 AP_RO_FAC_AVAIL

The AP_RO_FAC_AVAIL attribute is used to indicate the availability of facilities in the XAP-ROSE provider. The attribute is bit significant, and the following bit values are defined:

Name	Facility
AP_RO_BIND	BIND / UNBIND primitives

When a bit value is set in the attribute value the corresponding facility is available in the XAP-ROSE provider, otherwise the facility is unavailable.

3.2.2 AP_RO_PCI_LIST

The AP_RO_PCI_LIST attribute is used to define those abstract syntaxes that the ROSE service provider should examine for the presence of ROSE PDUs. The attribute consists of a list of presentation context identifiers (PCIs) for abstract syntaxes. These PCIs are used to reference the abstract syntaxes defined in the XAP environment variables AP_DCS, AP_PCDL or AP_PCDRL (depending on the current state of the XAP_ROSE instance).

A value assigned to the AP_RO_PCI_LIST environment attribute is validated only when the *ap_ro_init*( ) function is called. At this point, if the content of the list represents valid abstract syntaxes for the association, the list will be used by the ROSE service provider to send and receive ROSE PDUs. Subsequent changes to the value of the AP_RO_PCI_LIST attribute will not take effect until the *ap_ro_init*( ) function is called again.

3.3 Environment Attribute Summary Table

Attribute	Type/Values	Readable	Writable
AP_RO_PCI_LIST	ap_ro_pci_list_t default: none	always	always
AP_RO_FAC_AVAIL	unsigned long	always	never

3.4 Environment Attribute Typedefs

3.4.1 ap_ro_pci_list_t

This typedef is used to specify a list of abstract syntaxes that the ROPM is to examine to identify incoming ROSE PDUs. It is defined as:

```
typedef struct {
    int size_pcil,       /* Number of PCIs in list           */
    int *pci_list,       /* Pointer to an array of           */
                         /* presentation context identifiers */
} ap_ro_pci_list_t ;
```

Chapter 4

ROSE Functions

4.1 Introduction

4.1.1 Summary of ROSE Functions

The XAP-ROSE API provides the following functions in addition to those defined by the referenced XAP specifications:

- *ap_ro_init*( )
- *ap_ro_release*( )

4.1.2 Structure Definitions

XAP-ROSE is used in conjunction with XAP to establish an association with a remote ROSE service user and then pass ROSE primitives between the two ROSE service users. An include file for the XAP-ROSE user containing structure definitions and constants is defined in **<xap_rose.h>**.

The XAP-ROSE API defines an additional structure **ap_ro_cdata_t** that is pointed to by the *cdata* argument. This structure is used to pass ROSE-specific protocol information between the service user and the service provider. The **ap_ro_cdata_t** structure is defined as:

```
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"            */
  unsigned long     token_assignment; /* tokens assignment               */
  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                      */
  /*
   *  XAP-ROSE cdata
   */
  long            pci;                /* P. context id for user data     */
  long            priority;           /* Informative to provider,        */
                                      /* (optional)                      */
  long            invoke_id_present;  /* invoke id present flag          */
  long            invoke_id;          /* operation invocation identifier */
  long            linked_id_present;  /* linked id identifier present    */
  long            linked_id;          /* invocation identifier of        */
                                      /* parent operation                */
  long            op_class;           /* class of operation              */
  long            type;               /* value/result/operation          */
  union {
      unsigned long     local;
      ap_objid_t        global;
  } value;                            /* value of operation argument     */
} ap_ro_cdata_t;
```

Note: The ROSE-specific **ap_ro_cdata_t** structure elements *priority* and *op_class* support an RTSE-based implementation of ROSE which is currently not within the scope of this API. They are defined by this specification for future use and are not currently used.

NAME

ap_ro_init — initialise the ROSE user-environment.

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The *ap_ro_init*( ) function enables the operation of the ROSE service provider, allowing the API user to send XAP-ROSE request primitives and causing the service provider to filter incoming primitives to identify and deliver XAP-ROSE indication primitives.

If this function returns success, the ROSE service provider has been enabled. It then remains enabled until disabled by a call to *ap_ro_release*( ), or the XAP-ROSE instance is closed using a call to the XAP *ap_close*( ) function. When enabled, the ROSE service provider becomes active (that is, filters incoming PDU for the presence of ROSE PDUs) whenever an association is established, and is dormant at other times.

When this function is called, the list of ROSE abstract syntaxes (held in the AP_RO_PCI_LIST environment attribute) is validated to check that it contains only abstract syntaxes that are available or will be available, and that the negotiated transfer syntaxes are supportable by the XAP-ROSE service provider:

- If *ap_ro_init*( ) is called when an association exists, the list is validated against those abstract syntaxes that appear in the defined context set (held in the AP_DCS environment attribute).
- If *ap_ro_init*( ) is called before an association has been fully established, the list is validated against the proposed presentation context definition list (held in the AP_PCDL or AP_PCDRL environment attribute).

If this validation step fails, the function returns an error and the ROSE service provider is not enabled.

Arguments are as follows:

fd This integer value refers to the descriptor returned from a previous *ap_open*( ) call. It identifies the library instance that supports the association.

aperrno_p In case of failure, *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_ro_init*( ) returns 0. Otherwise, a value of –1 is returned and the ;ocation pointed to by *aperrno_p* is set to indicate the error.

ERRORS

[AP_RO_BAD_PCI]	The PCI is not in the AP_PCDL OR AP_DCS.
[AP_RO_EMPTY_LIST]	The list has no elements.
[AP_RO_CNTX_NOT_PRES]	One or more of the contexts identified is not in the relevant associations (presentation) defined context set.
[AP_RO_ILLEGAL_SIZE]	The list size is not a positive value or list is greater than the size of the AP_DCS attribute list.
[AP_RO_T_SYTX_NSUP]	The transfer syntax in the AP_DCS entry for this PCI is not supported.
[AP_NOT_SUPPORTED]	The identified instance is not an XAP-ROSE instance - that is, the service provider selected in the call to *ap_open*( ) does not support a ROSE service provider, or the AP_ROSE_MODE flag has not been set in the AP_MODE_SEL environment attribute.

SEE ALSO

ap_ro_release( ).

NAME

ap_ro_release — release the ROSE user-environment.

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The *ap_ro_release*( ) function releases an instance of the XAP-ROSE user environment from an established application_association.

If this function returns success, the ROSE service provider has been disabled, and will no longer be active when an application association is established.

fd This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

aperrno_p In case of failure, *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_ro_init*( ) returns 0. Otherwise, a value of –1 is returned and the ;ocation pointed to by *aperrno_p* is set to indicate the error.

ERRORS

[AP_NOT_SUPPORTED] The identified instance is not an XAP-ROSE instance - that is, the service provider selected in the call to *ap_open*( ) does not support a ROSE service provider, or the AP_ROSE_MODE flag has not been set in the AP_MODE_SEL environment attribute.

SEE ALSO

ap_ro_init( ).

Chapter 5

ROSE Primitives

This chapter presents *manual pages* for each of the primitives of the underlying OSI services to which XAP-ROSE provides access.

Each *man-page* provides a short description of an XAP-ROSE primitive, including the circumstances under which it may be used, followed by a detailed description of the parameters associated with it.

5.1 Summary of ROSE Primitives

The (XAP-ROSE) ROSE API includes a collection of ROSE primitives that let a user access the Remote Operation Protocol Machine (ROPM).

The valid list of ROSE primitives is:

- AP_RO_BIND_REQ
- AP_RO_BIND_IND
- AP_RO_BIND_RSP
- AP_RO_BIND_CNF
- AP_RO_INVOKE_REQ
- AP_RO_INVOKE_IND
- AP_RO_RESULT_REQ
- AP_RO_RESULT_IND
- AP_RO_ERROR_REQ
- AP_RO_ERROR_IND
- AP_RO_REJECTU_REQ
- AP_RO_REJECTU_IND
- AP_RO_REJECTP_IND
- AP_RO_UNBIND_REQ
- AP_RO_UNBIND_IND
- AP_RO_UNBIND_RSP
- AP_RO_UNBIND_CNF

NAME

AP_RO_BIND_REQ — initiate establishment of an application association

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_BIND_REQ primitive is used with *ap_snd*( ) and the XAP-ROSE environment to begin the establishment of an association between two application entities wishing to use ROSE services. After sending an AP_RO_BIND_REQ primitive, no other primitives can be issued, except A_ABORT_REQ until an AP_RO_BIND_CNF or A_PABORT_IND primitive is received.

The ROPM maps the AP_RO_BIND_REQ primitive to the A_ASSOC_REQ service directly. The effects and restrictions of sending the AP_RO_BIND_REQ primitive are identical to the A_ASSOC_REQ primitive.

Refer to the table on the *ap_snd*( ) manual page, under the A_ASSOC_REQ section, for these effects and restrictions.

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

fd: This integer value refers to the descriptor returned from a previous *ap_open*( ) call. It identifies the library instance that supports the association.

sptype: This argument must be set to AP_RO_BIND_REQ.

cdata: The following members of the cdata structure are used for this primitive:

```
long           udata_length; /* length of user-information */
long           pci;          /* Presentation Context Id    */
ap_a_assoc_env_t *env;       /* environment attribute      */
                             /* values                     */
```

The *cdata→udata_length* argument must be set to the number of octets of encoded user information that will be sent with this primitive if the primitive is issued as more than one *ap_snd*( ) invocation. If the primitive is issued as a single *ap_snd*( ) invocation, this field will be ignored.

The *cdata→pci* argument must be set to a value representing the presentation context id of a valid abstract syntax contained in the environment attribute AP_RO_PCI_LIST. The ROSE PDU will be encoded within a presentation data value identified by this *pci*.

The *cdata→env* argument can be used to override XAP environment attribute values used as parameters to the A-ASSOCIATE request which carries the RO-BIND 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 elements defined on the A_ASSOC_REQ manual page in the

XAP specification are used for this primitive.

ubuf: Use of the *ubuf* argument is described on the *ap_snd*( ) manual page.

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

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*( ) in the referenced **XAP** specification.

ERRORS

In addition to those listed in the manual page for *ap_snd*( ), the following AP_RO_BIND_REQ errors can occur:

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

[AP_BADCD_TOKENS] The value of tokens is not valid.

SEE ALSO

ro_intro, *ap_intro*, *ap_env*( ), *ap_open*( ), *ap_snd*( ), A_ASSOC_REQ.

NAME

AP_RO_BIND_IND - show a request to establish an association

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_BIND_IND primitive is used with *ap_rcv*( ) and the XAP-ROSE environment to show a request to establish an association between two application entities wishing to use ROSE services.

The ROPM maps the AP_RO_BIND_IND primitive from the A_ASSOC_IND service directly. The effects and restrictions of receiving the AP_RO_BIND_IND primitive are identical to the A_ASSOC_IND primitive.

Refer to the table on the *ap_rcv*( ) manual page, under the A_ASSOC_IND section, for these effects and restrictions.

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

fd: This integer value refers to the descriptor returned from a previous *ap_open*( ) call. It identifies the library instance that supports the association.

sptype: The value pointed to by this argument will be set to AP_RO_BIND_IND.

cdata: The following members of the cdata structure are used for this primitive:

```
long            udata_length; /* length of               */
                              /* user-information        */
long            pci;          /* Presentation Context ID */
ap_a_assoc_env_t *env;        /* environment attribute   */
                              /* values                  */
```

The *cdata→udata_length* argument will be set to show the total number of octets of encoded user-information received with this primitive.

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU.

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 which carries the RO-BIND 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 substructures by passing a pointer to *cdata* to *ap_free*( ). The A_ASSOC_IND

manual page in the XAP specification details the the elements of the **ap_a_assoc_env_t** structure used for this primitive.

ubuf: Use of the ubuf argument is described on the *ap_rcv*( ) manual page.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_rcv*( ) in the referenced **XAP** specification.

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*( ) in the referenced **XAP** specification.

ERRORS

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

SEE ALSO

ro_intro, *ap_intro*, *ap_env*( ), *ap_open*( ), *ap_snd*( ), A_ASSOC_IND.

NAME

AP_RO_BIND_RSP — respond to an association request indication

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_BIND_RSP primitive is used with *ap_snd*( ) and the XAP-ROSE environment to respond to an association establishment request between two application entities wishing to use ROSE services.

The ROPM maps the AP_RO_BIND_RSP primitive to the A_ASSOC_RSP service directly. The effects and restrictions of sending the AP_RO_BIND_RSP primitive are identical to the A_ASSOC_RSP primitive.

Refer to the table on the *ap_snd*( ) manual page, under the A_ASSOC_RSP section, for these effects and restrictions.

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

fd: This integer value refers to the descriptor returned from a previous *ap_open*( ) call. It identifies the library instance that supports the association.

sptype: This argument must be set to AP_RO_BIND_RSP.

cdata: The following members of the cdata structure are used for this primitive:

```
long           res;          /* Result of association   */
                             /* request                 */
long           diag;         /* Reason (if rejected)    */
long           udata_length; /* length of user          */
                             /* information             */
long           pci;          /* Presentation Context Id */
ap_a_assoc_env_t *env;       /* environment attribute   */
                             /* values                  */
```

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

- AP_ACCEPT
  accept the association
- AP_REJ_PERM
  association permanently rejected.

The *cdata→diag* argument is used to show the reason for the result specified by *res*. Refer to the manual page A_ASSOC_RSP *diag* argument for acceptable values.

The *cdata→udata_length* argument must be set to the number of octets of encoded user-information that will be sent with this primitive if the primitive is issued as more than one *ap_snd*( ) invocation. If the primitive is issued as a single *ap_snd*( ) invocation, this field will be ignored.

The *cdata→pci* argument must be set to a value representing the presentation context id of a valid abstract syntax contained in the environment attribute AP_RO_PCI_LIST. The ROSE PDU will be encoded within a presentation data value identified by this *pci*.

The *cdata→env* argument can be used to override XAP environment attribute values used as parameters to the A-ASSOCIATE response which carries the RO-BIND response 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 elements defined on the A_ASSOC_RSP manual page in the XAP specification are used for this primitive.

ubuf: Use of the *ubuf* argument is described on the *ap_snd*( ) manual page.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_snd*( ) in the referenced **XAP** specification.

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*( ) in the referenced **XAP** specification.

ERRORS

In addition to those listed in the manual page for *ap_snd*( ), the following error conitions are reported for this primitive:

[AP_BADSFU] An invalid combination of Session function units was proposed.

[AP_BADCD_RES] The value of *res* is not valid.

[AP_BADCD_DIAG] The value of *diag* is not valid.

[AP_BADCD_TOKENS] The value of *tokens* is not valid.

SEE ALSO

ro_intro, *ap_intro*, *ap_env*( ), *ap_intro*( ), *ap_open*( ), *ap_snd*( ), A_ASSOC_RSP.

NAME

AP_RO_BIND_CNF — confirm an association request

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_BIND_CNF primitive is used with *ap_rcv*( ) and the XAP-ROSE environment to confirm the establishment of an association between two application entities wishing to use ROSE services.

The ROPM maps the AP_RO_BIND_CNF primitive from the A_ASSOC_CNF service directly. The effects and restrictions of receiving the AP_RO_BIND_CNF primitive are identical to the A_ASSOC_CNF primitive.

Refer to the table on the *ap_snd*( ) manual page, under the A_ASSOC_CNF section, for these effects and restrictions.

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

fd: This integer value refers to the descriptor returned previous *ap_open*( ) call. It identifies the library instance that supports the association.

sptype: The value pointed to by this argument will be set to AP_RO_BIND_CNF.

cdata: The following members of the cdata structure are used for this primitive:

```
long           res;          /* Result of association   */
                             /* request                 */
long           res_src;      /* Source of result        */
long           diag;         /* Reason (if rejected)    */
long           udata_length; /* length of               */
                             /* user-information field  */
                             /* of APDU                 */
long           pci;          /* Presentation Context Id */
ap_a_assoc_env_t *env;       /* environment attribute   */
                             /* values                  */
```

The *cdata→res* argument will be set to show the result of the association request. The possible values for res are:

- AP_ACCEPT
  accept the association
- AP_REJ_PERM
  association permanently rejected.

The *cdata→res_src* argument shows the source of the result. The argument *diag* gives a diagnostic *code*. Values for these arguments are exactly as defined for the A_ASSOC_CNF primitive. Refer to A_ASSOC_CNF *res_src* and *diag* arguments for acceptable values.

The *cdata→udata_length* argument will be set to show the total number of octets of encoded user-information received with this primitive.

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU.

The *cdata→env* argument can be used to retrieve the values of the XAP environment attributes that correspond to parameters of the A-ASSOCIATE confirmation which carries the RO-BIND 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 substructures by passing a pointer to *cdata* to *ap_free*( ). The A_ASSOC_CNF manual page in the XAP specification details the the elements of the **ap_a_assoc_env_t** structure used for this primitive.

ubuf: Use of the *ubuf* argument is described on the *ap_rcv*( ) manual page.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_rcv*( ) in the referenced **XAP** specification.

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*( ) in the referenced **XAP** specification.

ERRORS

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

SEE ALSO

ro_intro, *ap_intro*, *ap_env*( ), *ap_open*( ), *ap_rcv*( ), A_ASSOC_CNF.

NAME

AP_RO_ERROR_REQ — used in response to an unsuccessfully performed operation

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_ERROR_REQ primitive is used with the *ap_snd*( ) function to let an application provide a negative result response to the remote application on negative outcome of an invoked operation.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: This argument must be set to AP_RO_ERROR_REQ

cdata: The following members of the cdata structure are used for this primitive:

```
long  udata_length;
long  pci;
long  priority;          /* provider opt                */
long  invoke_id;
long  type;              /* must set to AP_RO_LOCAL     */
                         /* or AP_RO_GLOBAL             */
union {
  unsigned long  local;  /* valid if type==AP_RO_LOCAL  */
  ap_objid_t     global; /* valid if type==AP_RO_GLOBAL */
} value;
```

The *cdata→pci* argument must be set to a value representing the presentation context id of a valid abstract syntax contained in the environment variable *ap_ro_pci_list*. The ROSE PDU will be encoded within a presentation data value identified by the *pci*.

Note: The ROSE-specific *cdata* structure element *priority* supports an RTSE-based implementation of ROSE which is currently not within the scope of this API. It is defined by this specification for future use and is not currently used.

The *cdata→type* argument signals the form the error value will be sent in. If *type* is set to AP_RO_LOCAL, the error value is in local format and is contained as an unsigned long in *value.local*. If *type* is set to AP_RO_GLOBAL, the error value is in global format.

The *global* element of the *cdata→value* element is an **ap_objid_t** structure containing the contents octets of the BER encoding of the OBJECT IDENTIFIER. See the referenced **XAP** specification and ISO 8825-1, Basic Encoding Rules (BER) for details of how to use this structure.

Where this primitive is to be sent using a series of calls to *ap_snd*( ), with the AP_MORE flag set, the *cdata→udata_length* element should be set to the total number of octets of encoded data, (representing the *error-parameter* parameter of the remote operation error response) that will be sent with this primitive. If the total number of octets of encoded data is not known, this field may be set to –1. The way in which this element is used and its possible effect on performance is described in the manual page for the P_DATA_REQ primitive in the referenced **XAP** specification.

ubuf: This function argument is used to pass a data value that represents the *error-parameter* parameter of the remote operation error response. The data value is in encoded form: *cdata→pci* identifies the presentation context used to encode it. This presentation context identifies both the abstract syntax (which defines the type of value encoded), and the transfer syntax (which defines how it was encoded). Use of the *ubuf* argument is described in the *ap_snd*( ) manual page in the referenced **XAP** specification.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_snd*( ) in the referenced **XAP** specification.

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

RETURN VALUE

Refer to *ap_snd*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_snd*( ) in the referenced **XAP** specification.

In addition to the errors listed in the *ap_snd*( ) manual page in the referenced **XAP** specification, the following error code has the defined additional meaning for XAP-ROSE primitives:

[AP_BADPRIM] For XAP-ROSE primitives, this error code may also be returned if the identified instance is not an XAP-ROSE instance - that is, the service provider selected in the call to *ap_open*( ) does not support a ROSE service provider, or the AP_ROSE_MODE flag has not been set in the AP_MODE_SEL environment attribute.

SEE ALSO

ap_snd( ) in the referenced **XAP** specification.

NAME

AP_RO_ERROR_IND — show the unsuccesful result of an invoked operation by the remote application.

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The **AP_RO_ERROR_IND** primitive is used with *ap_rcv*( ) and the XAP-ROSE environment to show the negative result of an unsuccessfully performed remote operation.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: The value pointed to will be set to AP_RO_ERROR_IND.

cdata: The following members of the cdata structure are used for this primitive:

```
long  pci;
long  invoke_id;
long  type;              /* set to AP_RO_LOCAL or    */
                         /* AP_RO_GLOBAL             */
                         /* or AP_RO_NO_RESULT       */
union {
  unsigned long  local;  /* valid if                 */
                         /* value_type==AP_RO_LOCAL  */
  ap_objid_t     global; /* valid if                 */
                         /* value_type==AP_RO_GLOBAL */
} value;
```

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU.

The *cdata→type* argument shows the form the error value was recieved in. If *type* is set to AP_RO_LOCAL, the error value is in local format and is contained as an unsigned long in *value.local*. If *type* is set to AP_RO_GLOBAL, the error value is in global format.

In global format, the error value will be contained in the **ap_objid_t** structure *value.global*. This global error value should be user encoded according to ISO 8825, Basic Encoding Rules (BER), as an object identifier.

ubuf: This function argument is used to return a data value that represents the *error-parameter* parameter of the remote operation error response. The data value is in encoded form: *cdata→pci* identifies the presentation context used to encode it. This presentation context identifies both the abstract syntax (which defines the type of value encoded), and the transfer syntax (which defines how it was encoded). Use of the *ubuf* argument is described in the *ap_rcv*( ) manual page in the referenced **XAP** specification.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_rcv*( ) in the referenced **XAP** specification.

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 *ap_rcv*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_rcv*( ) in the referenced **XAP** specification.

SEE ALSO

ap_rcv( ) in the referenced **XAP** specification.

NAME

AP_RO_INVOKE_REQ — request the start of a remote operation

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The AP_RO_INVOKE_REQ primitive is used with *ap_snd*( ) and the XAP-ROSE environment to request the start of a remote operation.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: This argument must be set to AP_RO_INVOKE_REQ.

cdata: The following members of the cdata structure are used for this primitive:

```
long  udata_length;
long  pci;
long  priority;          /* provider option             */
long  invoke_id;
long  linked_id_present; /* must be set TRUE or FALSE   */
long  linked_id;         /* invoke id of parent oper    */
long  op_class;          /* user optional               */
long  type;              /* must be set to AP_RO_LOCAL  */
                         /* or AP_RO_GLOBAL             */
union {
  unsigned long  local;  /* valid if                    */
                         /* value_type==AP_RO_LOCAL     */
  ap_objid_t     global; /* valid if                    */
                         /* value_type==AP_RO_GLOBAL    */
} value;
```

The *cdata→pci* argument must be set to a value representing the presentation context id of a valid abstract syntax contained in the environment variable *ap_ro_pci_list*. The ROSE PDU will be encoded within a presentation data value identified by the *pci*.

Note: The ROSE-specific *cdata* structure element *priority* and *op_class* support an RTSE-based implementation of ROSE which is currently not within the scope of this API. They are defined by this specification for future use and are not currently used.

The *cdata→type* argument signals the form the operation value will be sent in. If *type* is set to AP_RO_LOCAL, the operation value is in local format and is contained as an integer in *value.local*. If *type* is set to AP_RO_GLOBAL, the operation value is in global format.

In global format, the operation value must be contained in the **ap_objid_t** structure in *value.global*. This global operation value should be encoded according to ISO 8825, Basic Encoding Rules (BER), as an object identifier.

Where this primitive is to be sent using a series of calls to *ap_snd*( ), with the AP_MORE flag set, the *cdata→udata_length* element should be set to the total number of octets of encoded data, (representing the *argument* parameter of the remote operation invocation) that will be sent with this primitive. If the total number of octets of encoded data is not known, this field may be set to –1. The way in which this element is used and its possible effect on performance is described in the manual page of the P_DATA_REQ primitive in the referenced **XAP** specification.

ubuf: This function argument is used to pass a data value that represents the *argument* parameter of the remote operation invocation. The data value is in encoded form: *cdata→pci* identifies the presentation context used to encode it. This presentation context identifies both the abstract syntax (which defines the type of value encoded), and the transfer syntax (which defines how it was encoded). Use of the *ubuf* argument is described in the *ap_snd*( ) manual page in the referenced **XAP** specification.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_snd*( ) in the referenced **XAP** specification.

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

RETURN VALUE

Refer to *ap_snd*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_snd*( ) in the referenced **XAP** specification.

In addition to the errors listed in the *ap_snd*( ) manual page in the referenced **XAP** specification, the following error code has the defined additional meaning for XAP-ROSE primitives:

[AP_BADPRIM] For XAP-ROSE primitives, this error code may also be returned if the identified instance is not an XAP-ROSE instance - that is, the service provider selected in the call to *ap_open*( ) does not support a ROSE service provider, or the AP_ROSE_MODE flag has not been set in the AP_MODE_SEL environment attribute.

SEE ALSO

ap_snd( ) in the referenced **XAP** specification.

NAME

AP_RO_INVOKE_IND — show the start of an operation

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The AP_RO_INVOKE_IND primitive is used with *ap_rcv*( ) and the XAP-ROSE environment to show the start of an operation.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: The value pointed to will be set to AP_RO_INVOKE_IND.

cdata: The following members of the cdata structure are used for this primitive:

```
long  pci;
long  invoke_id;
long  linked_id_present; /* TRUE or FALSE            */
long  linked_id;         /* invoke id of parent oper */
long  type;              /* set to AP_RO_LOCAL or    */
                         /* AP_RO_GLOBAL             */
union {
  unsigned long  local;  /* valid if                 */
                         /* value_type==AP_RO_LOCAL  */
  ap_objid_t     global; /* valid if                 */
                         /* value_type==AP_RO_GLOBAL */
} value;
```

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU.

The *cdata→type* argument shows the form the operation value was recieved in. If *type* is set to AP_RO_LOCAL, the operation value is in local format and is contained as an unsigned long in *value.local*. If *type* is set to AP_RO_GLOBAL, the operation value is in global format.

In global format, the operation value will be contained in the ap_objid_t structure in value.global. This global operation value should be user encoded according to ISO 8825, Basic Encoding Rules (BER), as an object identifier.

The structure *cdata* and values for *op_class* are defined in the header file **<xap.h>**.

ubuf: This function argument is used to return a data value that represents the *argument* parameter of the remote operation invocation. The data value is in encoded form: *cdata→pci* identifies the presentation context used to encode it. This presentation context identifies both the abstract syntax (which defines the type of value encoded), and the transfer syntax (which defines how it was encoded). Use of the *ubuf* argument is described in the *ap_rcv*( ) manual page in the referenced **XAP** specification.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_rcv*( ) in the referenced **XAP** specification.

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 *ap_rcv*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_rcv*( ) in the referenced **XAP** specification.

SEE ALSO

ap_rcv( ) in the referenced **XAP** specification.

NAME

AP_RO_REJECTP_IND — show a problem has been detected by the ROSE provider.

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The AP_RO_REJECTP_IND primitive is used with *ap_rcv*( ) and XAP-ROSE environment to show the ROSE provider has detected a problem with a previous request. This request can also be identified by the returned invoke id.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: The value pointed to will be set to AP_RO_REJECTP_IND.

cdata: The following members of the cdata structure are used for this primitive:

```
long  pci;
long  invoke_id_present;    /* TRUE or FALSE */
long  invoke_id;
long  type;
long  rsn;
```

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU.

A reject code will include two parts, a *type*, and a *rsn*. *type* will be AP_RO_GENERAL_TYPE. Values for both *type* and *rsn* are defined in the header file **<xap_rose.h>**.

ubuf: Use of the *ubuf* argument is described on the **ap_rcv** manual page.

Returned parameter information is flagged by *rsn* == AP_RO_RETURN_PARM. Returned parameter information is always present when *rsn* == AP_RO_RETURN_PARM. Returned parameter information will never be present for other reject reasons.

The returned parameter information, if present, will contain the structure pointed to by the cdata of the primitive determined to be in error by the ROSE provider.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_rcv*( ) in the referenced **XAP** specification.

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 *ap_rcv*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_rcv*( ) in the referenced **XAP** specification.

SEE ALSO

ap_rcv( ) in the referenced **XAP** specification.

NAME

AP_RO_REJECTU_REQ — used by the ROSE user to reject an indication

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The AP_RO_REJECTU_REQ primitive is used with *ap_snd*( ) and the XAP-ROSE environment to reject a request (AP_RO_INVOKE_IND) or reply (AP_RO_RESULT_IND, AP_RO_ERROR_IND) from the remote ROSE user when problems are detected.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: This argument must be set to AP_RO_REJECTU_REQ.

cdata: The following members of the cdata structure are used for this primitive:

```
long  pci;
long  invoke_id_present;  /* must be set to TRUE */
                          /* or FALSE            */
long  invoke_id;
long  type;
long  rsn;
```

The *cdata→pci* argument must be set to a value representing the presentation context id of a valid abstract syntax contained in the environment variable *ap_ro_pci_list*. The ROSE PDU will be encoded within a presentation data value identified by the *pci*.

A reject code will include two parts, a *type*, and a *rsn*. *type* will be either AP_RO_INVOKE_TYPE, AP_RO_RESULT_TYPE or AP_RO_ERROR_TYPE. Values for both *type* and the *rsn* that are valid for each *type* are defined in the header file **<xap_rose.h>**.

ubuf: No user data is defined for this primitive, therefore *ubuf* should be set to NULL.

flags: No user data is defined for this primitive, therefore *flags* should be set to 0.

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

RETURN VALUE

Refer to *ap_snd*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_snd*( ) in the referenced **XAP** specification.

In addition to the errors listed in the *ap_snd*( ) manual page in the referenced **XAP** specification, the following error code has the defined additional meaning for XAP-ROSE primitives:

[AP_BADPRIM] For XAP-ROSE primitives, this error code may also be returned if the identified instance is not an XAP-ROSE instance - that is, the service provider selected in the call to *ap_open*( ) does not support a ROSE service provider, or the AP_ROSE_MODE flag has not been set in the AP_MODE_SEL environment attribute.

SEE ALSO

ap_snd( ) in the referenced **XAP** specification.

NAME

AP_RO_REJECTU_IND — show a problem has been detected by the remote ROSE user.

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The AP_RO_REJECTU_IND primitive is used with *ap_rcv*( ) and the XAP-ROSE environment to show the remote ROSE user has detected a problem with the previously submitted request (AP_RO_INVOKE_REQ) or reply (AP_RO_RESULT_REQ, AP_RO_ERROR_REQ). The request can also be identified by the returned invoke id.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: The value pointed to will be set to AP_RO_REJECTU_IND.

cdata: The following members of the cdata structure are used for this primitive:

```
long  pci;
long  invoke_id_present;    /* TRUE or FALSE */
long  invoke_id;
long  type;
long  rsn;
```

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU.

A reject code will include two parts, a *type*, and a *rsn*. *type* will be either AP_RO_INVOKE_TYPE, AP_RO_RESULT_TYPE, or AP_RO_ERROR_TYPE. Values for both *type* and the *rsn* that are valid for each *type* are defined in the header file **<xap_rose.h>**.

ubuf: No user data is defined for this primitive, therefore *ubuf* will remain unchanged.

flags: No user data is defined for this primitive, therefore *flags* will remain unchanged.

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 *ap_rcv*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_rcv*( ) in the referenced **XAP** specification.

SEE ALSO

ap_rcv( ) in the referenced **XAP** specification.

NAME

AP_RO_RESULT_REQ — reply to a previous AP_RO_INVOKE indication when an operation is performed successfully

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The AP_RO_RESULT_REQ primitive is used with the *ap_snd*( ) and the XAP-ROSE environment to show the AP_RO_RESULT service to the remote host. This service is in response to a successfully performed operation on behalf of the remote application.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: This argument must be set to AP_RO_RESULT_REQ.

cdata: The following members of the cdata structure are used for this primitive:

```
long  udata_length;
long  pci;
long  invoke_id;
long  type                /* AP_RO_LOCAL, AP_RO_GLOBAL */
                          /* or AP_RO_NO_RESULT        */
union {
  unsigned long  local;   /* valid if                  */
                          /* type == AP_RO_LOCAL       */
  ap_objid_t     global;  /* valid if                  */
                          /* type == AP_RO_GLOBAL      */
} value;
```

The *cdata→pci* argument must be set to a value representing the presentation context id of a valid abstract syntax contained in the environment variable *ap_ro_pci_list*. The ROSE PDU will be encoded within a presentation data value identified by the *pci*.

The *cdata→type* argument signals whether the optional fields of operation and result are sent and the form they take.

If type is set to NORESULT, the optional operation-value and result sequence will not be present in the ROSE APDU. This means that no user data may be supplied; any attempt to provide data results in the error code [AP_BADDATA] being returned. If *type* is set to AP_RO_LOCAL, the operation value is in local format and is contained as an unsigned long in *value.local*. If *type* is AP_RO_GLOBAL, the operation value is in global format.

In global format, the operation value must be contained in the ap_objid_t structure *value.global*. This global operation value should be encoded according to ISO 8825, Basic Encoding Rules (BER), as an object identifier.

Where this primitive is to be sent using a series of calls to *ap_snd*( ), the the AP_MORE flag set, the *cdata→udata_length* element should be set to the total number of octets of encoded data, (representing the *result* parameter of the remote operation result) that will be sent with this primitive. If the total number of octets of encoded data is not known, this field may be set to –1. The way in which this element is used and its possible effect on performance is described in the manual page of the P_DATA_REQ primitive in the referenced **XAP** specification.

The structure *cdata* is defined in the header file **<xap_rose.h>**.

ubuf: This function argument is used to pass a data value that represents the *result* parameter of the remote operation result. The data value is in encoded form: *cdata→pci* identifies the presentation context used to encode it. This presentation context identifies both the abstract syntax (which defines the type of value encoded), and the transfer syntax (which defines how it was encoded). Use of the *ubuf* argument is described in the *ap_snd*( ) manual page in the referenced **XAP** specification.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_snd*( ) in the referenced **XAP** specification.

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

RETURN VALUE

Refer to *ap_snd*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_snd*( ) in the referenced **XAP** specification.

In addition to the errors listed in the *ap_snd*( ) manual page in the referenced **XAP** specification, the following error code has the defined additional meaning for XAP-ROSE primitives:

[AP_BADPRIM] For XAP-ROSE primitives, this error code may also be returned if the identified instance is not an XAP-ROSE instance - that is, the service provider selected in the call to *ap_open*( ) does not support a ROSE service provider, or the AP_ROSE_MODE flag has not been set in the AP_MODE_SEL environment attribute.

SEE ALSO

ap_snd( ) in the referenced **XAP** specification.

NAME

AP_RO_RESULT_IND — show the receipt of positive results from a requested operation invocation.

SYNOPSIS

```
#include <xap_rose.h>

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

DESCRIPTION

The AP_RO_RESULT_IND primitive is used with *ap_rcv*( ) and the XAP-ROSE environment to show the successful result of a completed remote operation.

fd: This integer value refers to the descriptor returned from a previous *ap_open* call. It identifies the library instance that supports the association.

sptype: The value pointed to will be set to AP_RO_RESULT_IND.

cdata: The following members of the cdata structure are used for this primitive:

```
long  pci;
long  invoke_id;
long  type;               /* AP_RO_LOCAL, AP_RO_GLOBAL   */
                          /*  or AP_RO_NO_RESULT         */
union {
  unsigned long  local;   /* valid if type==AP_RO_LOCAL  */
  ap_objid_t     global;  /* valid if type==AP_RO_GLOBAL */
} value;
```

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU.

The *cdata→type* argument shows the form the optional operation and result fields were received in. If *type* is set to NO RESULT, no result was received, and the operation and result fields are undefined. If *type* is set to AP_RO_LOCAL, the operation was received in local format and is contained in *value.local*. If *type* is set to AP_RO_GLOBAL, the operation was received in global format. In global format, the operation value will be contained in the ap_objid_t structure in *value.local*. This global operation value should be user encoded according to ISO 8825, Basic Encoding Rules (BER), as an object identifier.

The structure *cdata* is defined in the header file **<xap_rose.h>**.

ubuf: This function argument is used to return a data value that represents the *result* parameter of the remote operation result. The data value is in encoded form: *cdata→pci* identifies the presentation context used to encode it. This presentation context identifies both the abstract syntax (which defines the type of value encoded), and the transfer syntax (which defines how it was encoded). Use of the *ubuf* argument is described in the *ap_rcv*( ) manual page in the referenced **XAP** specification.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_rcv*( ) in the referenced /B XAP specification.

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 *ap_rcv*( ) in the referenced **XAP** specification.

ERRORS

Refer to *ap_rcv*( ) in the referenced **XAP** specification.

SEE ALSO

ap_rcv( ) in the referenced **XAP** specification.

NAME

AP_RO_UNBIND_REQ - request the release an association

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_UNBIND_REQ primitive is used with *ap_snd*( ) and the XAP-ROSE environment to request the normal release of an association between two application entities using ROSE service.

The ROPM maps the AP_RO_UNBIND_REQ primitive to the A_RELEASE_REQ service directly. The effects and restrictions of sending the AP_RO_UNBIND_REQ primitive are identical to the A_RELEASE_REQ primitive.

Refer to the table on the *ap_snd*( ) manual page, under the A_RELEASE_REQ section, for these effects and restrictions.

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

fd: This integer value refers to the descriptor returned from a previous *ap_open*( ) call. It identifies the library instance that supports the association.

sptype: This argument must be set to AP_RO_UNBIND_REQ.

cdata: The following members of the cdata structure are used for this primitive:

```
long    rsn;             /* reason for release         */
long    udata_length;    /* length of user information */
long    pci;             /* Presentation Context Id    */
```

The *cdata→rsn* argument must be set to AP_REL_NORMAL.

The *cdata→udata_length* argument must be set to the number of octets of encoded user-information that will be sent with this primitive if the primitive is issued as more than one *ap_snd*( ) invocation. If the primitive is issued as a single *ap_snd*( ) invocation, this field will be ignored.

The *cdata→pci* argument must be set to a value representing the presentation context id of a valid abstract syntax contained in the environment attribute AP_RO_PCI_LIST. The ROSE PDU will be encoded within a presentation data value identified by this *pci*.

ubuf: Use of the *ubuf* argument is described on the *ap_snd*( ) manual page.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_snd*( ) in the referenced **XAP** specification.

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*( ) in the referenced **XAP** specification.

ERRORS

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

In addition to those listed in the manual page for *ap_snd*( ), the following AP_RO_UNBIND_REQ errors can occur:

[AP_BADTOKEN] The token(s) required to issue this primitive are not currently held.

[AP_BADCD_RSN] The value of *rsn* is not valid.

SEE ALSO

ro_intro, *ap_intro*, *ap_env*( ), *ap_open*( ), *ap_snd*( ), A_RELEASE_REQ.

NAME

AP_RO_UNBIND_IND — show an association release request

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_UNBIND_IND primitive is used with *ap_rcv*( ) and the XAP-ROSE environment to show the remote service user wants to release the association which uses ROSE services.

The ROPM maps the AP_RO_UNBIND_IND primitive from the A_RELEASE_IND service directly. The effects and restrictions of receiving the AP_RO_UNBIND_IND primitive are identical to the A_RELEASE_IND primitive.

Refer to the table on the *ap_snd*( ) manual page under the A_RELEASE_IND section for these effects and restrictions.

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

fd: This integer value refers to the descriptor returned from a previous *ap_open*( ) call. It identifies the library instance that supports the association.

sptype: The value pointed to by this argument will be set to AP_RO_UNBIND_IND.

cdata: The following members of the cdata structure are used for this primitive:

```
long    rsn;             /* reason for release         */
long    udata_length;    /* length of user information */
long    pci;             /* Presentation Context Id    */
```

The *cdata→rsn* argument will be set to AP_REL_NORMAL.

The *cdata→udata_length* argument will be set to show the total number of octets of encoded user information that were received with this primitive.

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU. If no user information was recieved, the value of *pci* will be undefined.

ubuf: Use of the *ubuf* argument is described on the *ap_snd*( ) manual page.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_rcv*( ) in the referenced **XAP** specification.

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*( ) in the referenced **XAP** specification.

ERRORS

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

SEE ALSO

ro_intro, *ap_intro*, *ap_env*( ), *ap_open*( ), *ap_rcv*( ), A_RELEASE_IND.

NAME

AP_RO_UNBIND_RSP — respond to an association release request

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_UNBIND_RSP primitive is used with *ap_snd*( ) and the XAP-ROSE environment to respond to an association release request of an association using ROSE services.

The ROPM maps the AP_RO_UNBIND_RSP primitive to the A_RELEASE_RSP service directly. The effects and restrictions of sending the AP_RO_UNBIND_RSP primitive are identical to the A_RELEASE_RSP primitive.

Refer to the table on the *ap_snd*( ) manual page under the A_RELEASE_RSP section for these effects and restrictions.

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

fd: This integer value refers to the descriptor returned from a previous *ap_open*( ) call. It identifies the library instance that supports the association.

sptype: This argument must be set to AP_RO_UNBIND_RSP.

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

```
long    res;            /* result                      */
long    rsn;            /* reason for the result       */
long    udata_length;   /* length of user-information */
long    pci;            /* Presentation Context Id     */
```

The *cdata→res* argument must be set to AP_REL_AFFIRM.

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

- AP_REL_NORMAL
  Shows a normal release
- AP_REL_NOTFINISHED
  Shows the rejection of the release because the user is not finished with the association.

The *cdata→udata_length* argument must be set to the number of octets of encoded user-information that will be sent with this primitive if the primitive is issued as more than one *ap_snd*( ) invocation. If the primitive is issued as a single *ap_snd*( ) invocation, this field will be ignored.

The *cdata→pci* argument must be set to a value representing the presentation context id of a valid abstract syntax contained in the environment attribute AP_RO_PCI_LIST. The ROSE PDU will be encoded within a presentation data value identified by this *pci*.

ubuf: Use of the ubuf argument is described on the *ap_snd*( ) manual page.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_snd*( ) in the referenced **XAP** specification.

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*( ) in the referenced **XAP** specification.

ERRORS

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

In addition to those listed in the manual page for *ap_snd*( ), the following errors can occur:

[AP_BADCD_RES] The value of *res* is not valid

[AP_BADCD_RSN] The value of *rsn* is not valid.

SEE ALSO

ro_intro, *ap_intro*, *ap_env*( ), *ap_open*( ), *ap_snd*( ), A_RELEASE_RSP.

NAME

AP_RO_UNBIND_CNF — confirm a release request

SYNOPSIS

```
#include    <xap_rose.h>

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

DESCRIPTION

The AP_RO_UNBIND_CNF primitive is used with *ap_rcv*( ) and the XAP-ROSE environment to confirm the acceptance or rejection of a previously sent release request of an association using ROSE services.

The ROPM maps the AP_RO_UNBIND_CNF primitive from the A_RELEASE_CNF service directly. The effects and restrictions of receiving the AP_RO_UNBIND_CNF primitive are identical to the A_RELEASE_CNF primitive.

Refer to the table on the *ap_snd*( ) manual page under the A_RELEASE_CNF section for these effects and restrictions.

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

fd: This integer value refers to the descriptor returned from a previous ap_open call. It identifies the library instance that supports the association.

sptype: The value pointed to by this argument will be set to AP_RO_UNBIND_CNF.

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

```
long    res;          /* Result of the release request */
long    rsn;          /* Reason for the result         */
long    udata_length; /* length of user information    */
long    pci;          /* Presentation Context Id       */
```

The *cdata→res* argument will be set to AP_REL_AFFIRM.

The reason associated with the result will be shown by *rsn*. The possible values for *rsn* are as follows:

- AP_REL_NORMAL
  Shows a normal release.
- AP_REL_NOTFINISHED
  Shows the rejection of the release because the remote user was not finished with the association.

The *cdata→udata_length* argument will be set to show the total number of octets of encoded user-information received with this primitive.

The *cdata→pci* argument will be set to the value of the presentation context id encoded within the presentation data value which contained the ROSE PDU. If no user information was received, the value of *pci* will be undefined.

ubuf: Use of the *ubuf* argument is described on the *ap_rcv*( ) manual page.

flags: This argument is used to control certain aspects of primitive processing as described in the manual page for *ap_rcv*( ) in the referenced **XAP** specification.

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*( ) in the referenced **XAP** specification.

ERRORS

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

SEE ALSO

ro_intro, *ap_intro*, *ap_env*( ), *ap_open*( ), *ap_rcv*( ), A_RELEASE_CNF.

Appendix A

XAP-ROSE Header File

This Appendix reproduces the contents of the <xap_rose.h> header file for the applications which wish to use the XAP-ROSE API.

```
/*
 *  xap_rose.h
 *
 *  Contains structures and constants needed to interface
 *  with the ROSE protocol machine through XAP.
 *
 */

#ifndef AP_ROSE_ID
#define AP_ROSE_ID (13)

#ifndef AP_ID
#include <xap.h>

#endif

/*
 *  XAP-ROSE flag for AP_MODE_SEL
 */
#define AP_ROSE_MODE 0x04

/*
 *  Possible values for cdata->type argument
 */
#define AP_RO_LOCAL        1
#define AP_RO_GLOBAL       2
#define AP_RO_NO_RESULT    3

/*
 *  Flag for returned parameter information
 */
#define AP_RO_RETURN_PARM  1

/*
 *  Reject code type values
 */
#define AP_RO_INVOKE_TYPE  1
#define AP_RO_RESULT_TYPE  2
#define AP_RO_ERROR_TYPE   3

/*
 *  Primitive types
 */
#define AP_RO_INVOKE_IND    ((AP_ROSE_ID<<16) | 0x01)
```

```
#define AP_RO_INVOKE_REQ    ((AP_ROSE_ID<<16) | 0x02)

#define AP_RO_RESULT_IND    ((AP_ROSE_ID<<16) | 0x03)
#define AP_RO_RESULT_REQ    ((AP_ROSE_ID<<16) | 0x04)

#define AP_RO_ERROR_IND     ((AP_ROSE_ID<<16) | 0x05)
#define AP_RO_ERROR_REQ     ((AP_ROSE_ID<<16) | 0x06)

#define AP_RO_REJECTU_IND   ((AP_ROSE_ID<<16) | 0x07)
#define AP_RO_REJECTU_REQ   ((AP_ROSE_ID<<16) | 0x08)
#define AP_RO_REJECTP_IND   ((AP_ROSE_ID<<16) | 0x09)

#define AP_RO_BIND_REQ      ((AP_ROSE_ID<<16) | 0x0A)
#define AP_RO_BIND_IND      ((AP_ROSE_ID<<16) | 0x0B)
#define AP_RO_BIND_RSP      ((AP_ROSE_ID<<16) | 0x0C)
#define AP_RO_BIND_CNF      ((AP_ROSE_ID<<16) | 0x0D)

#define AP_RO_UNBIND_REQ    ((AP_ROSE_ID<<16) | 0x0E)
#define AP_RO_UNBIND_IND    ((AP_ROSE_ID<<16) | 0x0F)
#define AP_RO_UNBIND_RSP    ((AP_ROSE_ID<<16) | 0x10)
#define AP_RO_UNBIND_CNF    ((AP_ROSE_ID<<16) | 0x11)

/*
 *  The following are provider primitive types
 *  included here for value allocation purposes
 */
#define AP_RO_INFO_REQ      ((AP_ROSE_ID<<16) | 0x12)
#define AP_RO_INFO_ACK      ((AP_ROSE_ID<<16) | 0x13)
#define AP_RO_INFO_ACK_XAP  ((AP_ROSE_ID<<16) | 0x17)

/*
 *  The following are the ROSE specific error that
 *  the ROSE provider can return.
 */
#define AP_RO_ILLEGAL_SIZE  ((AP_ROSE_ID<<16) | 0x14)
#define AP_RO_EMPTY_LIST    ((AP_ROSE_ID<<16) | 0x15)
#define AP_RO_CNTX_NOT_PRES ((AP_ROSE_ID<<16) | 0x16)
#define AP_RO_BAD_PCI       ((AP_ROSE_ID<<16) | 0X18)
#define AP_RO_T_SYTX_NSUP   ((AP_ROSE_ID<<16) | 0X19)

/*
 *  Attribute identifiers
 */
#define AP_RO_FAC_AVAIL     ((AP_ROSE_ID<<16) | 0x01)
#define AP_RO_PCI_LIST      ((AP_ROSE_ID<<16) | 0x02)

/*
 *  Identifiers for use with ap_free()
 */
#define AP_RO_PCI_LIST_T    AP_RO_PCI_LIST
#define AP_RO_CDATA_T       ((AP_ROSE_ID<<16) | 0x03)
```

```
/*
 *  Bit masks for AP_RO_FAC_AVAIL
 */
#define AP_RO_BIND          (1<<0)

/*
 *  Environment Attribute Structure definitions special to ROSE
 */

typedef               /* Abstract syntaxes containing ROSE PDUs */
    struct {
    int    size_pcil;           /* Number of PCIs in list      */
    int     *pci_list;          /* Pointer to an array of PCIs */
} ap_ro_pci_list_t;

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"     */
    unsigned long token_assignment; /* tokens assignment       */
    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               */
/*
 *  XAP-ROSE cdata elements
 */
    long pci;               /* P. context id for user data     */
    long priority;          /* Informative to provider,        */
                            /* (optional)                      */
    long invoke_id_present; /* invoke id present flag          */
    long invoke_id;         /* operation invocation identifier */
    long linked_id_present; /* linked id identifier present    */
    long linked_id;         /* invocation identifier of        */
                            /* parent operation                */
    long class;             /* class of operation              */
    long type;              /* value/result/operation          */
```

```
    union {
        unsigned long  local;
        ap_objid_t     global;
    } value;                /* value of operation argument     */

} ap_ro_cdata_t;

#endif                      /* end AP_ROSE_ID                  */
```

Glossary

application-association
Defined by the OSI Basic Reference Model as ‘‘A cooperative relationship between two application-entity-invocations for the purpose of communication of information and coordination of their joint operation.’’

AE
Application Entity.

application-entity
Defined by the OSI Basic Reference Model 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 to implement the application.

APDU
Application-Protocol-Data-Unit

ASE
Application Service Element

application-service-element
Defined by the OSI Basic Reference Model 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 (for example, ACSE, which provides connection management services to be used by all applications), while others provide services oriented to a particular application (for example, FTAM, which provides file tranfer, access and management services).

ACSE
Association Control Service Element

child-operation
An operation which might be invoked by the *performer* of a *parent-operation* as part of a *linked-operation*

invoker
The *application-entity* that invokes a remote operation.

linked-operation
A set of operations consisting of a *parent-operation* and one or more *child-operations*

operation
A request by one entity (the *invoker*), which another entity (the *performer)* attempts to perform and then reports the outcome of the attempt.

operation-interface
The (conceptual) interface between a *user-element* and *ROSE-user ASE*s. The interface is defined as a set of remote operations using the RO-Notation.

parent-operation
An operation, during the operation of which, the *performer* may invoke zero or more *child-operations* as part of a *linked-operation*.

performer
The application-entity that attempts to perform a remote operation that has been invoked by the other *application-entity* .

RO
Remote Operations

ROPM
Remote Operation Protocol State Machine

ROSE
Remote Operation Service Elements

ROSE-user ASE
The application-specific function that performs the mapping of the operations, results and errors of a set of remote operations onto ROSE.

ROSPI
Remote Operation Service Provider Interface

RTSE
Reliable Transfer Service Element

user-element
Defined by the OSI Basic Reference Model as ‘‘The representation of that part of the application-process which uses those application-service-elements needed to accomplish the communications objectives of that application-process’’

XAP
ACSE / Presentation Library Interface

XAP-ROSE
Remote Operation Service Library Interface

Index