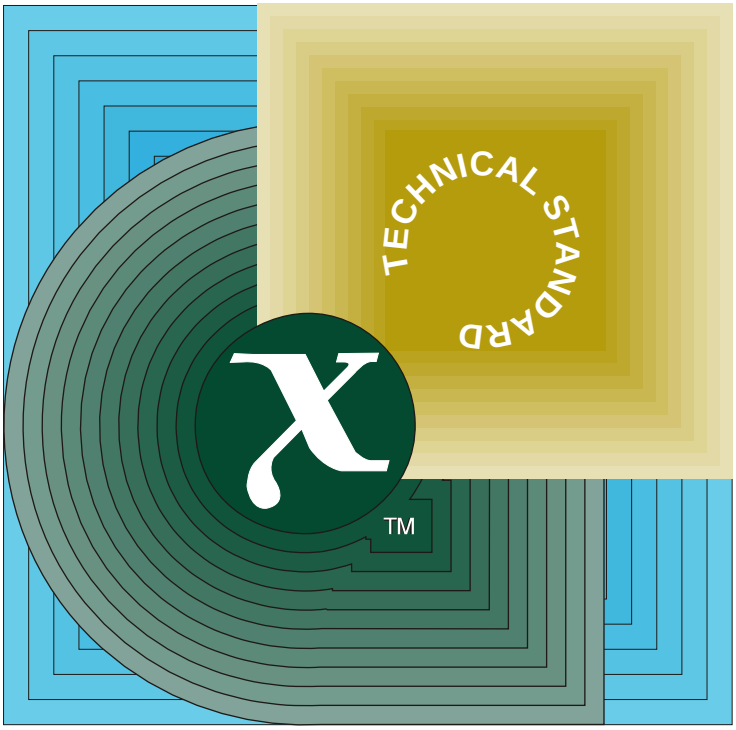
Technical Standard

ACSE/Presentation:
Transaction Processing API (XAP-TP)



THE Open GROUP

[This page intentionally left blank]

X/Open CAE Specification

ACSE/Presentation: Transaction Processing API (XAP-TP)

X/Open Company Ltd.

X/Open CAE Specification

ACSE/Presentation: Transaction Processing API (XAP-TP)

ISBN: 1-85912-091-1

X/Open Document Number: C409

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.org

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 a CAE Specification (see above). It defines the X/Open ACSE/Presentation (XAP) programming interface Transaction Processing extension (XAP-TP).

X/Open has already defined an ACSE/Presentation (XAP) programming interface (see **Referenced Documents**), which provides for access to the ISO OSI protocol stack at the upper two layers (Association Control Service Element and Presentation) of the OSI 7-layer model.

The XAP-TP API is an interface to the OSI Transaction Processing Service Element, and to the Presentation Layer, of the 7-layer Open Systems Interconnection model. It allows concurrent use of the existing XAP facilities on XAP instances supporting associations, in parallel with the use of XAP-TP facilities on XAP instances supporting OSI TP dialogues.

The XAP-TP extension to XAP provides a programming interface to the OSI Transaction Processing facilities, which support X/Open's interface specifications for TP applications (Peer-to-Peer, XATMI, TxRPC). In addition, the programming interface is at such a level as also to enable existing Transaction Monitors and applications to have access to OSI TP for interworking with new applications, thereby enabling users to retain their considerable investment in existing TP applications while they develop new X/Open-conformant ones.

Structure

- Chapter 1 explains the motivation for developing this XAP-TP API specification, and gives a short description of the services to which it provides access. It also defines the requirements placed upon implementations of this Systems Programming Interface (SPI). These include the minimum set of functions that must be provided by an implementation of XAP-TP, and the requirements placed upon the underlying implementation of the OSI TP and ACSE application-service-elements and Presentation Layer. Finally, it identifies areas of development of the OSI standards upon which this document are based, which may result in changes to the interface in a possible future version of this XAP-TP specification.
- Chapter 2 lists XAP-TP functions, and describes how they may be used to set up dialogues and transfer data. This chapter also describes the extensions to the structures provided for exchanging data and control information between XAP and the user, and shows how they are used.

- Chapter 3 defines the XAP-TP environment and the structures used for exchanging data and control information with the SPI.
- Chapter 4 presents the manual page definitions for the XAP-TP SPI. These define the functions which make up XAP-TP, giving the detailed specifications of parameters and data structures where there are differences from the corresponding XAP functions. Manual pages which have not changed from the XAP specification are not repeated here.
- Chapter 5 presents the manual page definition for the XAP-TP *xap_tp_osic* command.
- Chapter 6 provides information on the format of files used by XAP-TP. Specifically, it describes the XAP-TP environment file, which is used by the *xap_tp_osic* command.
- Chapter 7 presents manual page definitions for each of the primitives of the underlying OSI services to which the XAP provides access via the *ap_snd*( ) and *ap_rcv*( ) functions. Each manual page provides a short description of an OSI TP, ACSE or Presentation Layer primitive, including the circumstances under which it may be sent or received, and a detailed description of the parameters associated with it.
- Appendix A presents an example XAP-TP header file **<xap_tp.h>**.

Intended Audience

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

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

Applications Implementors
: Implementors of OSI applications and application-specific service APIs which are to run over OSI TP protocol stacks, may use this specification in conjunction with the appropriate application-specific protocol specifications to design applications and ASEs which are portable across OSI TP protocol stack implementations from different vendors running on different systems and/or different hardware (for example, X/Open TP-CRMs, RDA, OSI-RPC, and so on).

  Note: As a System Programming Interface, XAP-TP is not expected to be used directly by end users for the implementation of end-user applications. Rather it is expected to be used for the construction of system-level components which may themselves be of direct use to end-user application developers.

Positioning

This specification provides an extension to the **XAP** specification. It is intended to be read in conjunction with that specification and so does not repeat text from XAP where it is unchanged from the XAP version.

The reader is expected to be familiar with the first three chapters of the **XAP** specification before reading this specification.

Throughout the rest of this specification, functions and primitives that are unchanged from XAP have been omitted — the reader should refer to the **XAP** specification for their definition.

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 [ABCD] 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

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

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

Referenced Documents

The following documents are referenced in this guide:

The OSI Reference Model

ISO 7498
ISO 7498: 1984, Information Processing Systems — Open Systems Interconnection — Basic Reference Model.

ISO ACSE

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 Presentation

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

ISO 8822, Amendment 5
ISO 8822: 1988, Information Processing Systems — Open Systems Interconnection — Connection-oriented Presentation Service Definition — Amendment 5: Additional Synchronization Facility.

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

ISO 8823, Amendment 5
ISO 8823: 1988, Information Processing Systems — Open Systems Interconnection — Connection-oriented Presentation Protocol Specification — Amendment 5: Additional Synchronization Facility..

ASN.1 Notation

ISO 8824
ISO 8824: 1990, Information Technology — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1).

ASN.1 Basic Encoding Rules

BER
: ISO/IEC 8825:1990 (ITU-T Recommendation X.209 (1988)), Information Technology — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).

OSI Session Layer

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

ISO 8326, Amendment 4
: ISO 8326: 1987, Information Processing Systems — Open Systems Interconnection — Basic Connection-oriented Session Service Definition — Amendment 4: Additional Synchronization Facility.

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

OSI Distributed Transaction Processing

OSI TP Model
: ISO/IEC 10026-1: 1992, Information Technology — Open Systems Interconnection — Distributed Transaction Processing — Part 1: OSI TP Model.

OSI TP Service
: ISO/IEC 10026-2: 1992, Information Technology — Open Systems Interconnection — Distributed Transaction Processing — Part 2: OSI TP Service.

OSI TP Protocol
: ISO/IEC 10026-3: 1992 Information Technology — Open Systems Interconnection — Distributed Transaction Processing — Part 3: Protocol Specification.

ISP 12061
: ISP 12061, Information Technology — Open System Interconnection — International Standardized Profiles: OSI Distributed Transaction Processing, Part 5 (ATP11), Part 6 (ATP12), Part 7 (ATP21), Part 8 (ATP22), Part 9 (ATP31), Part 10 (ATP32).

OSI Commitment, Concurrency and Recovery

ISO/IEC 9804
: ISO/IEC 9804: 1990, Information Technology — Open Systems Interconnection — Service Definition for the Commitment, Concurrency, and Recovery Service Element, together with:

  Technical Corrigendum 1: 1991 to ISO/IEC 9804: 1990
  Amendment 2: 1992 to ISO/IEC 9804: 1990 Session mapping changes.

ISO/IEC 9805
: ISO/IEC 9805: 1990, Information Technology — Open Systems Interconnection — Protocol Specification for the Commitment, Concurrency, and Recovery Service Element, together with:

Technical Corrigendum 1: 1991 to ISO / IEC 9805: 1990
Technical Corrigendum 2: 1992 to ISO / IEC 9805: 1990
Amendment 2: 1992 to ISO / IEC 9805: 1990 Session mapping changes.

X/Open Specifications

DTP

X / Open Guide, November 1993, Distributed Transaction Processing: Reference Model, Version 2 (ISBN: 1-85912-019-9, G307).

XAP

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

CPI-C Version 2

X / Open Preliminary Specification, November 1994, The CPI-C Specification, Version 2, X / Open Document Number P415, ISBN: 1-85912-057-1.

TxRPC

X / Open Preliminary Specification, July 1993, Distributed Transaction Processing: The TxRPC Specification (ISBN: 1-85912-000-8, P305).

XATMI

X / Open Preliminary Specification, July 1993, Distributed Transaction Processing: The XATMI Specification (ISBN: 1-872630-99-5, P306).

Chapter 1

Introduction

1.1 Motivation

X/Open has already defined a Systems Programming Interface (SPI) which can be used to access the OSI protocol stack at the ACSE and Presentation layers: the X/Open ACSE/Presentation programming interface (XAP).

This provides for the portability of OSI applications (for example, X.400, File Transfer Access and Management (FTAM), Directory Services, Network Management, VTP) to protocol stacks provided by multiple vendors by provision of a standardised interface to *core* common upper-layer functions.

X/Open is defining APIs for distributed transaction processing applications. The relationship between this Systems Programming Interface and the X/Open Distributed Transaction Processing (DTP) APIs is shown in Section 2.1.6 on page 27.

There are a number of reasons why X/Open has decided to extend this Systems Programming Interface to provide access to the services provided by OSI TP:

- OSI TP constrains the U-ASE used with it. It has rules for how U-ASE pdus are concatenated with OSI TP protocol data units (pdus). To control this concatenation, OSI TP must have knowledge of the P_DATA requests issued by the U-ASE. This is difficult to achieve with ASEs sharing use of an XAP instance.
- Many companies that implement or purchase OSI stacks providing OSI TP access do not wish to implement those OSI ASEs and applications which make use of it as they emerge. A standard interface capable of supporting these applications (such as X/Open DTP CRMs, RDA, OSI RPC, and so on) makes it possible for system vendors and users to purchase products from various independent software vendors (ISVs) and add them to their existing common upper-layer stack.
- In the TP world, considerable investment of time and effort has been expended producing current applications and Transaction Monitors. Users expect these programs to fulfill their purpose for their planned lifespan.

X/Open is currently producing interface specifications for TP applications. These will certainly be the basis for the majority of the new TP applications developed in the next decade; however, customers will continue to require that their existing applications operate until they reach the end of their useful life and their replacements have been designed and built.

During the transition period (which will be years), users will increasingly require their existing applications to interoperate with the new applications being built, until evolution causes the old applications and systems to be replaced by new X/Open-conformant systems and applications.

These existing applications and systems should not be excluded from interworking with new applications utilising OSI TP.

The XAP-TP provides a standardised interface to OSI TP at a level that enables existing Transaction Monitors and applications to have access to OSI TP for interworking with the new applications during this transition period.

For these reasons, X/Open has identified an extension to the **XAP** specification to provide access to the OSI TP ASE, whilst still providing access to the existing XAP facilities.

In the XAP specification the case for standardisation of the XAP interface has been clearly stated. The addition of OSI TP makes it possible for system vendors and users to purchase application-layer products utilising OSI TP from various ISVs. In addition, it allows the ISVs to design products to a single standard interface avoiding the investment needed to operate over multiple different OSI TP interfaces.

Figure 1-1 illustrates possible System Programming Interfaces to OSI services, and their relationship to potential applications.

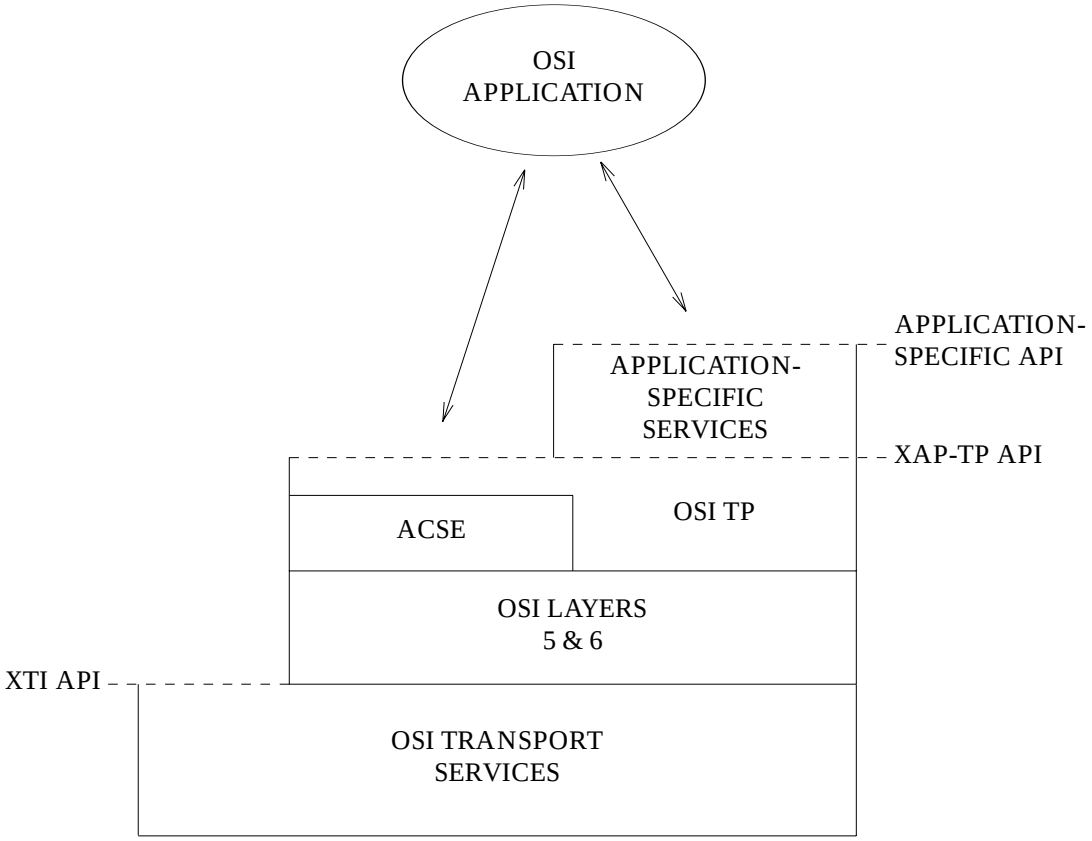


Figure 1-1 OSI Service Interfaces

The characteristics of the XAP-TP OSI TP interface allow:

- multiple simultaneous use by different Transaction Managers and ASEs
- existing applications being upgraded to support transactional semantics.

1.2 Scope and Purpose

The purpose of this document is to describe the extensions to the XAP Systems Programming Interface (SPI) for OSI TP support, and to define the new functions and data structures which it provides for use by applications. The specification defines the items of state information which control the operation of the SPI and its underlying *service-provider*, the states in which the primitives provided by the SPI are valid, and the effect on the state information of each of these primitives.

It is *not* the purpose of this specification to define a particular subset of the OSI TP, ACSE and Presentation Layer protocols which implementations must support. The compliance requirements for an implementation of the SPI and the underlying protocol implementation to which it provides access are defined in Section 1.5 on page 15.

1.3 Overview of OSI TP Services

In the remainder of this specification, readers are assumed to be familiar with the services provided by the ACSE and Presentation protocols. Those readers who require a brief overview of the services of the upper layers of the OSI protocol stack will find one in the equivalent overview section of the **XAP** specification.

This section provides a brief overview of the services of the OSI Transaction Processing Standard, for the benefit of those readers who require it. The text includes references to the OSI specifications for the services and protocols under discussion.

1.3.1 OSI Distributed Transaction Processing

This provides support for distributed, coordinated transaction processing. The OSI TP model is defined in ISO / IEC 10026-1: 1992 (the OSI TP Model), the service in ISO / IEC 10026-2: 1992 (the OSI TP Service), and the protocol in ISO / IEC 10026-3: 1992 (the OSI TP Protocol).

OSI TP introduces the concept of a TP Service User (TPSU) which is separately addressable within an Application Entity Invocation (AEI) by its TPSU-Title. Use of a TPSU running within an AEI is a TPSU Invocation (TPSUI).

These TPSUIs communicate with each other as peers. An instance of communication between TPSUIs is a dialogue. Within a dialogue, application-specific protocols are performed by the U-ASE. They are complemented by OSI TP protocols to perform transaction processing functions. Dialogues are established over associations, and OSI TP envisages pools of associations to minimise the association establishment costs.

The services provided by OSI TP are split into separate functional units, which are selected per dialogue at dialogue establishment time. The Dialogue functional unit provides the core capability of beginning and ending dialogues and reporting errors (TP-BEGIN-DIALOGUE, TP-END-DIALOGUE, TP-U-ERROR, TP-U-ABORT and TP-P-ABORT).

1.3.2 Polarised and Shared Control Functional Units

Dialogues may be operated in polarised control (half duplex mode), or in shared control (full duplex mode). In polarised control, only the TPSUI which has control of a dialogue may send data (by means of the U-ASE) and may issue commitment requests. In shared control both TPSUIs have control of the dialogue all the time.

Use of Polarised Control is enabled by selection of the Polarised Control functional unit. Similarly use of shared control is enabled by selection of the Shared Control functional unit. These functional units are mutually exclusive.

In polarised control, *turn* is passed using the TP-GRANT-CONTROL service, and can be requested using the TP-REQUEST-CONTROL service. Control may also be exchanged by rollback or by use of the TP-DEFERRED-GRANT-CONTROL service. These services are automatically available when the Polarised Control functional unit is selected. In shared control there is no need for these turn control services, and they are unavailable.

1.3.3 Handshake Functional Unit

In simple usage of polarised control the two TPSUIs using a dialogue can synchronise their processing by inspection of the received PDU stream. In more complex usage of polarised control and in shared control this is more difficult, and synchronisation services become necessary.

The Handshake services allow the synchronisation of two TPSUIs. They may be used when either the shared control or the polarised control is selected. When the shared control is selected both TPSUIs may request a synchronisation at the same time. Use of handshake is enabled by selection of the Handshake functional unit.

OSI TP provides combination TP-HANDSHAKE-AND-GRANT-CONTROL services for use when both the Polarised Control and Handshake functional units are selected, thus allowing optimisation of the protocol flows.

1.3.4 Commit Functional Unit

The OSI TP Commit functional unit provides the direct support for coordinated commitment of multiple dialogues. The OSI TP protocol uses the CCR ASE (Commitment Concurrency and Recovery — ISO / IEC 9804: 1990 and ISO / IEC 9805: 1990) to effect the coordination and recovery of transactions. OSI TP and CCR use a two-phase commit protocol, whereby a node brings its resources to a READY state, in which it must guarantee to be capable of releasing them in their final state if the transaction is committed, or in their initial state if the transaction is rolled back. (Final state is the state of the resources at the end of the transaction; initial state is the state of the resources at the beginning of the transaction.) When all resources involved in a transaction have reached the ready state, the initiator of the entire transaction issues an instruction to commit. If, for some reason, one of the resources cannot be brought to the ready state, this failure is reported and a rollback is triggered in the transaction tree.

This coordination of multiple resources is performed by the MACF in OSI TP (Multiple Association Control Facility). It coordinates the CCR flows on the multiple dialogues from the TPSUI to subordinate TPSUIs and the flows on the dialogue to the superior TPSUI (if one exists).

Until reaching the ready state, both OSI TP and the TPSUIs presume rollback. If a dialogue to a node is lost before the node has reached the ready state, the transaction shall be rolled back, and so no recovery is needed.

Note: When the Commit functional unit is selected, either the Chained transactions functional unit (see Section 1.3.12 on page 9) or the Unchained Transactions functional unit (see Section 1.3.11 on page 8) must also be selected.

1.3.5 Application Supported Transactions

When the TPSUIs communicating over a dialogue do not make use of the Commit functional unit, the TP Service Provider (TPSP) is not involved in any transaction or its commitment — if this functionality is needed then it must be performed by some private protocol that flows over the dialogue. The TPSP merely provides access to data transfer, error notification and dialogue control functionality. This is known as an *Application Supported Transaction*.

1.3.6 Provider Supported Transactions

When the TPSUIs communicating over a dialogue use the Commit functional unit, the TPSP provides direct support for the delineation of transactions and their commitment or rollback. This is known as a *Provider Supported Transaction*.

In the rest of this document *transaction* is taken to mean *provider supported transaction*, and *application supported transaction* is used explicitly where necessary.

Note: When both the Commit and Unchained Transactions functional units are selected on a dialogue but the dialogue is not currently involved in a (provider supported) transaction, it is possible to run an application supported transaction over it.

1.3.7 Dialogue Tree

In OSI TP dialogues are formed into trees, the dialogues linking nodes in the tree. The nodes are individual TPSUIs and the dialogues are the *branches* of the dialogue tree.

Within a dialogue tree the node that initiates a dialogue is referred to as the direct superior of the node with which the dialogue is established. The node with which the dialogue is established is referred to as the direct subordinate of the node that initiated the dialogue.

A node which has no superior is called a *root* node, one which has no subordinates is called a *leaf* node, and one which has both a superior and one or more subordinates is called an *intermediate* node.

This is shown in Figure 1-2.

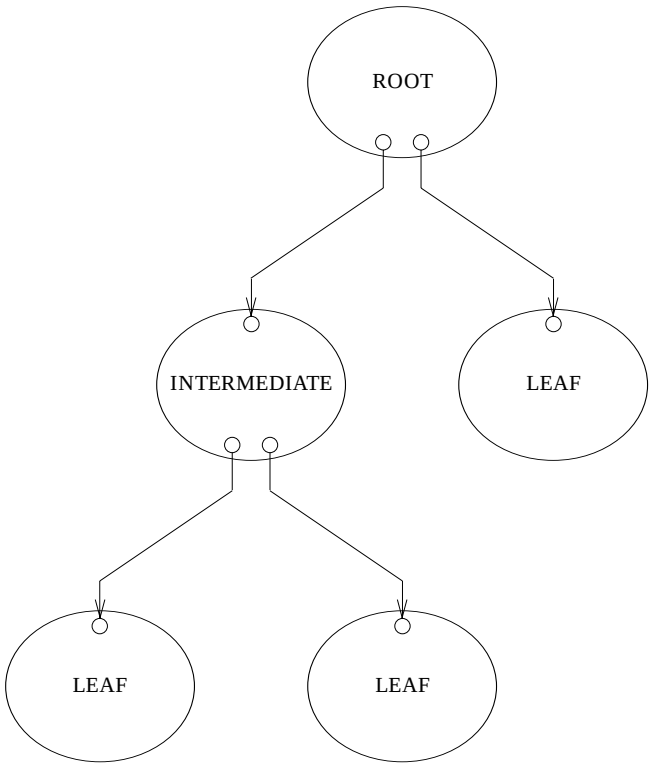


Figure 1-2 A Dialogue Tree

1.3.8 Transaction Branches

When a dialogue branch is running within a transaction (either application or provider supported) it is known as a transaction branch. In the OSI TP specification, and throughout the rest of this specification, *transaction branch* is used to refer to *provider supported transaction branch* and *application supported transaction branch* is used explicitly where necessary.

A dialogue supporting a transaction branch is called a coordinated dialogue (coordination-level = COMMIT). A dialogue not supporting a transaction branch is uncoordinated (coordination level = NONE).

Note: The superior always starts transaction branches to the subordinate. It is not possible in the current version of OSI TP for a subordinate in the dialogue tree to commence a transaction branch to its superior.

A new transaction branch is commenced either explicitly or implicitly:

- explicitly, by starting a dialogue in transaction mode or by issuing a TP-BEGIN-TRANSACTION request over an existing dialogue
- implicitly, by commitment or rollback of the current transaction branch when the Chained Transactions functional unit is selected (see Section 1.3.12 on page 9).

A transaction branch is terminated when the corresponding transaction is terminated (committed or rolled back).

1.3.9 Transaction Tree

In a similar way to the dialogue tree, the transaction tree is a tree with TPSUIs as nodes and (provider supported) transaction branches linking them. The same terms of *root*, *intermediate* and *leaf* are used to refer to the equivalent parts of the transaction tree. All the transaction branches are involved in a single distributed transaction, and the root node (together with its TPPM) is the coordinator of commitment for this transaction.

All the nodes and transaction branches of a transaction tree are involved in the same provider supported transaction.

There may be more than one transaction tree on a dialogue tree at any time; however, OSI TP constrains them to be separated by at least one uncoordinated dialogue branch (see Figure 1-3).

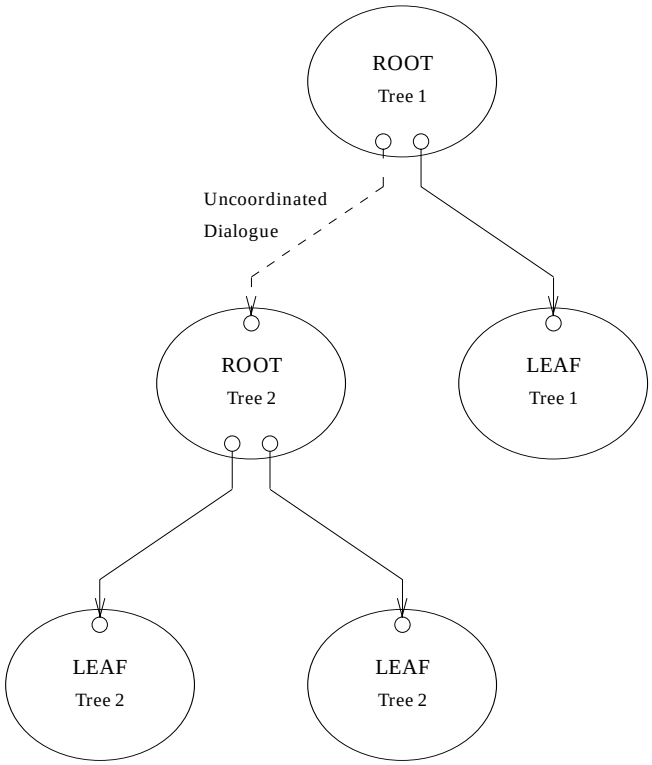


Figure 1-3 Two Transaction Trees on a Dialogue Tree

1.3.10 Relationship of the MACF to the TPSUI and Dialogue Tree

The dialogue from a superior and those dialogues to subordinates for a particular node (TPSUI) all share a single OSI TP MACF. In this way, it is guaranteed that a node (and those transaction branches related to it) will only participate in at most a single transaction at a time.

1.3.11 Unchained Transactions

If the Unchained Transactions functional unit is selected, transaction branches are explicitly started by issuing a TP-BEGIN-TRANSACTION request on an existing dialogue, or by starting a dialogue in transaction mode. A transaction branch is ended by commitment or rollback of the transaction, at which time the dialogue is free to be used for an application-supported transaction branch or to become part of another distributed transaction by issuing a TP-BEGIN-TRANSACTION request.

Note: When the Unchained Transactions functional unit is selected, the Commit functional unit must also be selected.

1.3.12 Chained Transactions

When the Chained Transactions functional unit is selected, the first transaction branch commences when the dialogue is established, and each time a transaction is committed or rolled back, a new transaction branch commences on the dialogue. The dialogue is always involved in a provider supported transaction, and so cannot be used for an application-supported transaction branch.

Note: When the Chained Transactions functional unit is selected the Commit functional unit must also be selected.

1.3.13 Mixing Chained and Unchained Transactions within a TPSUI

It is entirely permissible for a TPSUI to have a mixture of chained and unchained dialogues at the same time.

1.3.14 Neither Chained nor Unchained Transactions

If neither the Chained or Unchained Transactions functional unit is selected, the Commit functional unit shall not be selected.

1.3.15 Dialogues without Commit FU

Dialogues without the Commit functional unit selected may only participate in application-supported transactions.

1.3.16 Deferred End Dialogue

Because a dialogue with the Chained Transactions functional unit selected is always involved in a provider-supported transaction, it is not possible to utilise the TP-END-DIALOGUE request service. The TP-DEFERRED-END-DIALOGUE request service fills this role. A superior uses deferred end dialogue to instruct the TPPM to end the dialogue to a subordinate after commitment of the current transaction branch. A TP-DEFERRED-END-DIALOGUE indication is issued to the subordinate TPSUI to inform it that the dialogue will end after commitment. If the transaction branch is rolled back, the deferred end dialogue is cancelled and the dialogue placed in a new transaction.

The TP-DEFERRED-END-DIALOGUE request service may also be used when the Unchained Transactions functional unit is selected. This allows the caller to optimise the protocol flows across the underlying association. If the transaction is rolled back, the deferred end dialogue is cancelled and the dialogue continues but is no longer within a transaction.

Deferred End Dialogue is only available on a dialogue that supports a transaction branch operated in polarised control.

1.3.17 Deferred Grant Control

Similar to deferred end dialogue, the TP-DEFERRED-GRANT-CONTROL request service allows the superior to instruct the TPPM that control of the dialogue resides with the subordinate after commitment of the transaction branch. This is signalled to the subordinate TPSUI with a TP-DEFERRED-GRANT-CONTROL indication service.

Deferred Grant Control is only available on a dialogue operated in polarised control which supports a transaction branch.

1.3.18 Position of Control after Commitment or Rollback

On those dialogues using polarised control, when a transaction branch is committed, control of the dialogue resides with the superior unless a deferred grant control has been issued, in which case control resides with the subordinate.

If a transaction branch is rolled back control of the dialogue resides with the TPSUI which had control of the dialogue at the start of the transaction.

1.3.19 Atomic Action Identifiers

OSI TP uses CCR Atomic Action Identifiers (AAIds) to identify uniquely a provider-supported distributed transaction. The AAId applies to the entire transaction tree. So, each transaction branch of the tree has the *same* AAId. The AAId is formed from the Application Entity Title (AET) of the originator of the transaction and a suffix unique within the scope of the AET.

Note: The AAId must be globally unique.

1.3.20 Branch Identifiers

Each branch of a transaction tree has a branch identifier, unique within the scope of the AAID, consisting of the AET of the originator of the branch (the superior) and a suffix.

The combination of AAId and Branch Identifier (BRId) uniquely identifies an individual branch of a transaction, and is used by OSI TP on all CCR exchanges relating to that branch. Figure 1-4 shows the AAId and BRIds in use for a transaction tree.

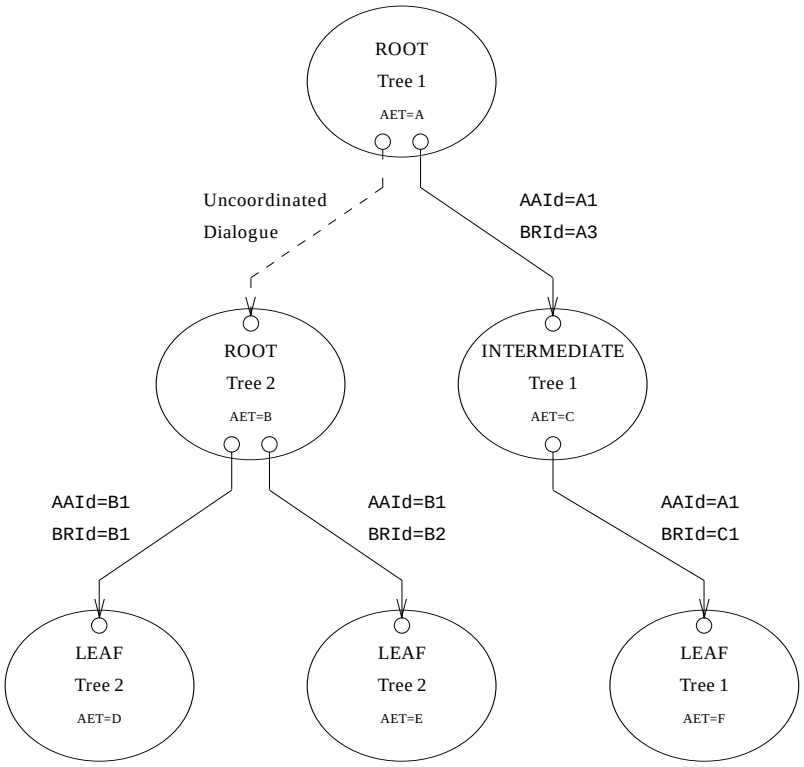


Figure 1-4 AAIds and BRIds in the Transaction Tree

1.3.21 Logging

In order to ensure recovery after a system node crash, OSI TP requires log records to be written at appropriate points.

OSI TP defines four different log records:

- **log-ready** record
- **log-commit** record
- **log-damage** record
- **log-heuristic** record.

These records have the following uses:

log-ready record
: This is secured prior to reporting *ready* to the superior and contains enough detail of the transaction node and its branches to enable a node to contact its subordinates (if any) and its superior during recovery.

log-commit record
: This is secured at a root node after all subordinates have reported *ready* and before the root node propagates the commit decision to its subordinates and the TPSUI. This ensures the root node can contact its subordinates during recovery.

log-damage record
: This holds details of a possible or actual heuristic condition for reporting to the TPSUI and the superior (if any). It contains a status of *heuristic-hazard* or *heuristic-mix* and enough detail to identify the transaction node and enable it to contact its superior (if any) during recovery. It is secured when one of the following is true:

  — the node has not reached the ready state and it loses contact with a subordinate after issuing a prepare PDU

  — the node receives the decision from the superior and determines that its bound data is inconsistent with the outcome of the transaction, and that it cannot rectify the situation

  — the node receives the outcome of the commit or rollback from a subordinate which reports that a heuristic damage condition has occurred.

  Reports from subordinates may result in a **log-damage** record being updated from *heuristic-hazard* to *heuristic-mix*.

log-heuristic record
: This is used to record the state of the bound data for a node when the node takes a heuristic decision. It contains details of the node, the state of the bound data, and information to enable the node to contact its superior (if any) during recovery.

Both **log-damage** and **log-heuristic** records persist until removed by local administrative means.

1.3.22 Recovery Context Handles

OSI TP provides the Recovery Context Handle mechanism to allow the identification of a grouping of log records to be used for all transactions on a particular association. Its use is optional — if it is not present on the association, no grouping is used. If used, the Recovery Context Handle is passed during recovery to identify the grouping.

1.3.23 Recovery

OSI TP only performs recovery for provider supported transaction branches. Recovery is initiated only by nodes that are either in the ready state or nodes that have propagated a commit decision. A node that is not in the ready or decided (commit) state automatically initiates a rollback in case of failure. Recovery is performed by OSI TP for the users *behind the scenes*, and so is not directly visible (see Section 8.7 of ISO / IEC 10026-1: 1992 (the OSI TP Model)). The terms *subordinate* and *superior* used during the discussion on recovery refer to the OSI TP recovery facilities at the subordinate and the superior nodes, respectively.

There are three possibilities:

- recovery from association loss
- recovery from process loss
- recovery from system crash.

These are described in the following three sections.

1.3.24 Recovery after Association Loss

If the TPSUI has not completed the first phase of the two phase commit, the transaction is rolled back and the node has no recovery responsibility.

The actions taken differ for the subordinate of the transaction branch and the superior as follows.

Subordinate

When an association with the superior is lost after the subordinate has reported (and logged) READY, but before the commit or rollback request has been received, the subordinate is *in doubt* and must determine the outcome of the transaction. The subordinate allocates a TP channel for recovery and issues a C-RECOVER(READY). The superior returns one of the following responses:

COMMIT — Commit the transaction branch; the subordinate shall commit the transaction in its subtree, forget the transaction branch and issue a C-RECOVER(DONE).

UNKNOWN — The transaction branch has been rolled back at the superior's side; the subordinate shall rollback the transaction in its subtree and forget the transaction branch.

RETRY LATER — The recover request cannot be handled at this time; the subordinate has to reissue C-RECOVER(READY) at a later time (it may also receive a C-RECOVER(COMMIT) from the superior).

When the association with the superior is lost after the subordinate has received a commit order, it finishes the commitment procedure in its subtree and then forgets the transaction branch; the subordinate is not in doubt and has no recovery responsibility. The subordinate then receives a C-RECOVER(COMMIT) and has to respond C-RECOVER(DONE) after the commitment procedure has been finished and the transaction branch forgotten.

When the association with the superior is lost while the subordinate is performing rollback, it finishes the rollback procedure in its subtree and forgets the transaction. No recovery is performed for this branch by either partner.

When a subordinate receives a C-RECOVER(COMMIT) for a transaction branch that it does not know, it replies C-RECOVER(DONE).

If a heuristic decision or a heuristic hazard exists in the subordinate subtree, the transaction branch is never forgotten (unless explicitly required by local means) and the heuristic report is conveyed to the superior in the C-RECOVER(DONE).

Superior

When an association with a subordinate is lost after the superior has issued a commit order, the superior allocates a TP channel for recovery and issues a C-RECOVER(COMMIT) on this channel. A response of DONE indicates that the commitment has been successfully performed in the subordinate's subtree (unless the C-RECOVER(DONE) contains a heuristic report).

When an association with a subordinate is lost while the superior is performing a rollback, it finishes the rollback procedure and forgets the transaction. No recovery is then performed for this branch by either partner. If a prepare has been sent to this subordinate, a heuristic hazard condition exits for this transaction branch and is logged. This is defined in Annex H of ISO / IEC 10026-3: 1992 (the OSI TP Protocol).

1.3.25 Recovery after Process Crash

OSI TP detects the loss of the process supporting the TPSUI. If this occurs prior to the completion of the first phase of the two phase commit, then the dialogues are aborted and any local resources for the transaction branch are rolled back and the branch forgotten.

If this occurs after the completion of the first phase of commit, then the action taken depends on the effect of the process loss on the OSI TP implementation, as follows:

- The process loss may cause loss of associations supporting the TPSUI. OSI TP treats this as described in Section 1.3.24 on page 12.
- The process loss may result in the loss of the entire TPPM. OSI TP reconstitutes the TPPM and then treats the case as association loss (see Section 1.3.24 on page 12).
- The process loss causes neither loss of associations nor loss of the TPPM. OSI TP continues the commitment or rollback of the transaction branch without the involvement of the TPSUI.

1.3.26 Recovery after System Crash

This is seen by the subordinates and superiors of TPSUIs affected as association loss. On reload of the system, the TPPMs are reconstructed from log data, and recovery occurs for each transaction branch as described in Section 1.3.24 on page 12.

1.4 Terminology

Definition of Terms

The terminology of this specification is that defined in the OSI standards to which XAP-TP provides access. For convenience, the **Glossary** provides brief definitions of many of these terms. This specification is one of a number of extensions to the **XAP** specification, and as such utilises the terminology and conventions of that specification.

Use of Naming Prefixes

In order to preserve uniqueness, all functions, typedefs, data items and constants unique to this document have names that begin with the prefix *ap_tp_* or AP_TP_. Functions, typedefs, data items and constants used within this specification but defined by the **XAP** specification have names that begin with the prefix *ap_* or AP_ as required by that specification.

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

Alignment with ISO C

The definition of this API has been modified to align it with the ISO C standard:

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

1.5 XAP-TP Compliance

All the XAP functions, as detailed in the **XAP Compliance** section of the **XAP** specification must be provided. However, the functionality defined for some of these functions is optional. When a function call is made to a service which is not available, the implementation must return the error code [AP_NOT_SUPPORTED].

The choice of underlying profile, as reflected in the appropriate Protocol Implementation Conformance Statement (PICS) for the ISO/IEC, determines which XAP-TP Service primitives, listed in Table 2-1 on page 29 and Table 2-2 on page 30, are supported. This information must be given in the conformance statement submitted for branding purposes.

International Standardized Profiles for OSI TP are currently under review within ISO. These profiles include reference information to be used while completing PICSs. PICS proformas for the ISO/IEC Session, Presentation, ACSE, CCR and TP protocol specifications are currently under review.

An implementation which complies with this specification shall be conformant to one or more of the profiles ATP11, ATP12, ATP21, ATP22, ATP31 and ATP32, which are defined in ISP 12061 (TP Profiles). Which of these profiles are supported has to be stated in the conformance statement submitted for branding purposes.

Conformance to the OSI TP profiles requires conformance to ISO/IEC Session, Presentation, ACSE, CCR and OSI TP, as specified in ISO/IEC 10026-3:1992 (the OSI TP Protocol). Conformant completion of the related PICS proformas shall also apply to implementations claiming conformance to this specification, once these PICS have been approved by ISO/IEC.

1.6 Future Directions

Work is progressing within ISO in the following areas:

- commitment optimisations (including read-only, one-phase commit, rollback diagnostics, migration of commit coordinator, dynamic commitment)
- subtransactions
- dialogue recovery
- separation of data and commit flows (that is, on different associations).

This specification will be updated to reflect this work when it becomes standardised.

In addition, any changes deemed necessary as a result of pilot implementations of this Preliminary Specification will be taken into account.

Chapter 2

Overview of XAP-TP

This chapter provides an overview of the XAP-TP extension to the XAP interface, its relationship to the X/Open DTP and OSI TP models, its functions, and the mechanisms provided for communication with it. A brief example of how the interface may be used to establish a dialogue and transfer data is also provided.

Readers are expected to have read Chapter 1 prior to reading this chapter.

2.1 XAP-TP Model

XAP-TP uses the Service User, Service Provider, XAP Instance, XAP Environment, Service Primitive Parameters and User Data features of the **XAP** specification unchanged.

The TP extension introduces new environment attributes: an extended cdata structure (**ap_tp_cdata_t**) for use with *ap_snd*( ) and *ap_rcv*( ); and further primitives which can be utilised when *TP_MODE* is selected for an XAP instance.

XAP-TP provides access to two categories of service primitives:

- dialogue primitives, which apply to an individual dialogue
- control primitives, which apply to all transaction mode dialogues of one TPSUI.

XAP-TP allows the user to map these two categories to one or more processes (see Section 2.5 on page 34).

The following sections describe the X/Open DTP Model, the OSI TP Model, their relationship to each other, and their relationship to XAP-TP. They discuss the relationship between the models. They *do not* provide an implementation guide, nor do they provide an exhaustive description of all aspects of the models.

2.1.1 X/Open DTP Model

The X/Open DTP model describes 3 components: an Application Program (AP), a Transaction manager (TM), and one or more Resource managers (RMs). (See the the **DTP** guide.) The DTP model without communications is as shown in Figure 2-1.

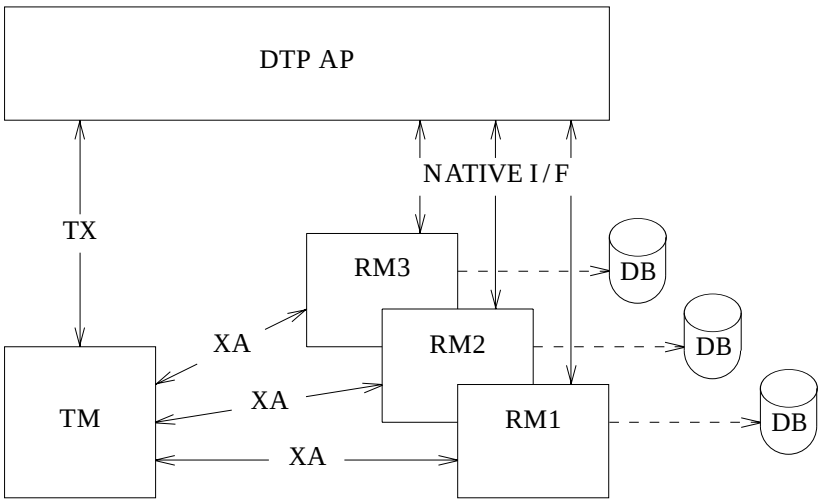


Figure 2-1 DTP Model without Communications

A DTP Application uses the Transaction demarcation interface (TX) to delineate the start and end of transactions.

An RM controls access to a resource (for example a database), and a DTP application uses the native interface provided by the RM (for example, SQL) to manipulate the resource.

The TM uses the XA interface to control the delineation, commitment and rollback of transactions with the RMs. XA provides two phase commit, which enables the TM to coordinate the commitment of multiple RMs, ensuring that resources are kept in a consistent state.

During the first phase of commitment, each RM logs its READY status to secure storage prior to reporting READY to the TM. This enables recovery, to a consistent state, after process or system crash.

The relationship between the TM and RMs is analogous to that in OSI TP between the MACF and the SACFs of individual dialogues.

An RM may provide the view of a single resource to the DTP Application and the TM whilst employing multiple resources internally. In this case, the RM coordinates resources internally to provide this view. For example, a database may consist internally of multiple file sections on multiple discs.

To sum up:

- The TM coordinates the preparation, commitment and rollback of one (or more) resources (represented by RMs) in use by the application.
- The TX interface is used for the application to request commitment or rollback of the TM, and for the TM to report the outcome of the commitment or rollback to the application.
- The TM uses the XA interface to coordinate the RMs.
- The RMs are responsible for logging their transaction details to secure storage.
- An RM may provide a single resource by internally coordinating multiple resources.

2.1.2 X/Open DTP Model with Communications

The introduction of communications changes the DTP model, and requires an extended interface between a Communication RM (CRM) and the TM. This extended interface is XA+.

XA is extended in two ways to make XA+:

- New Log primitives allow any RM to pass responsibility for logging to the TM.
- Additional two phase commit primitives allow a CRM to control the two phases of the commitment of the TMs' subordinate resources.

When communication is in use there are two cases to consider:

- The *root* of the transaction is the DTP AP. Commitment of the transaction is requested locally by the DTP AP, and is controlled by the TM. The TM acts as superior to all the RMs, including those providing communication to subordinates nodes.
- The *root* of the transaction lies elsewhere. Commitment of the transaction locally is controlled by the CRM providing the communications with the superior node. The DTP AP and TM are subordinate to the CRM. The TM still acts as superior to the other RMs in use.

The first case, where DTP application is the ROOT node of the transaction, is straightforward. This behaves as described above in Section 2.1.1 on page 18, with the exception that CRMs can exploit the XA+ feature to pass log records to the TM for logging to secure storage. This case is not discussed further here.

The second case, where the ROOT node of the transaction resides elsewhere, is more complex. An example of the X/Open DTP model with communications from a superior is shown in Figure 2-2 on page 20.

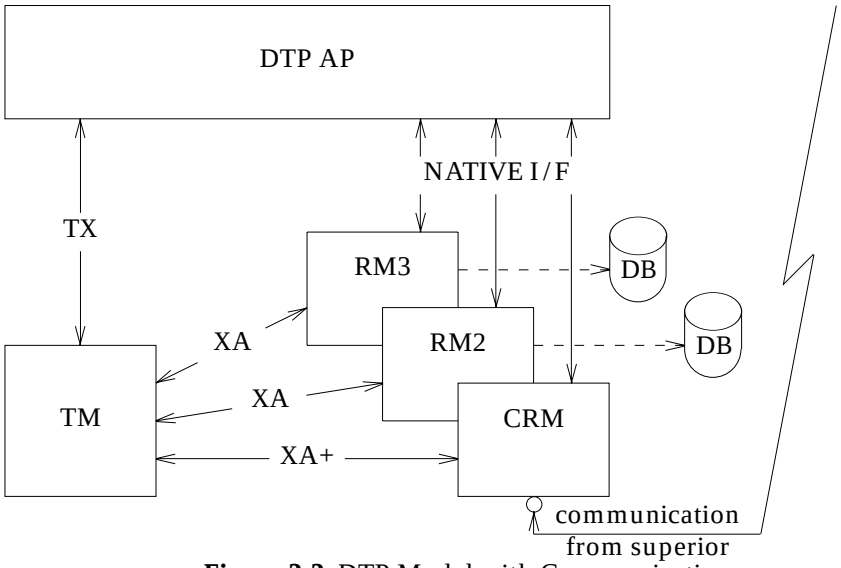


Figure 2-2 DTP Model with Communications

When a DTP AP becomes a subordinate in a transaction, by communication from a superior node through a CRM, the DTP AP can no longer request overall commitment of the transaction, it can only indicate to the TM its readiness to participate in commitment of the transaction. The CRM explicitly controls the two phases of commitment with exchanges to the TM through the XA+ interface.

The first phase of commitment (the prepare phase) is entered when the subordinate node receives a TP-PREPARE-RI pdu from its superior. Processing of the commit procedure is continued after the DTP AP has signalled its willingness to participate in commitment by calling *tx_commit*( ) — which it does either as a result of communication received from its superior node through the CRM, or as a result of a local decision that its part in the transaction is ready for commitment.

Once the TM has received the *tx_commit*( ) request from the DTP application it instructs the RMs to bring their resources into the READY state. The TM reports the outcome of preparation to the CRM (and so to the superior) using XA+. If one (or more) of the RMs fail to reach the READY state, the TM rolls back the transaction locally and informs the CRM of the rollback. The CRM informs the superior which then rolls back the rest of the transaction.

The CRMs may pass log information to the TM for inclusion in its transaction log through the XA+ interface. The destination and source of all the transaction-mode communications are recorded to enable recovery after process or system crash.

After the TM has informed the superior (through the CRM) of successfully reaching the READY state, the CRM (using the XA+ interface) informs the TM of the decision by the node to commit or rollback the transaction. The TM reports the result of commitment (or rollback) to the CRM through the XA+ interface.

To sum up:

- XA+ allows the CRM to pass responsibility for logging to the TM.
- XA+ enables a CRM to act as superior to the TM (instead of the subordinate role RMs play in XA) when the ROOT node of the transaction is elsewhere.

2.1.3 OSI TP Model

The OSI TP model describes application-supported transactions. They are not discussed further here, because their delineation, commitment, rollback and recovery are performed without any protocol involvement by OSI TP.

The OSI TP model describes a distributed transaction as a series of interconnected NODES forming a transaction tree. The OSI TP model for a node in the transaction tree is shown in Figure 2-3.

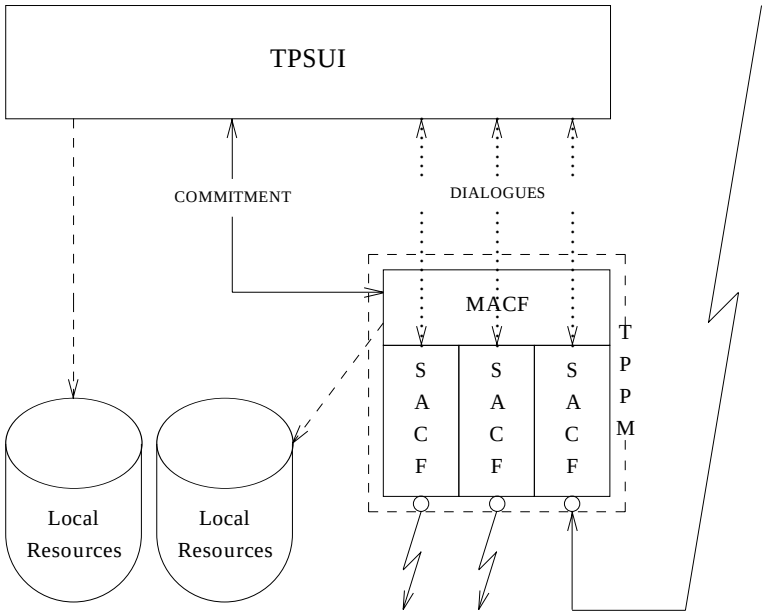


Figure 2-3 OSI TP Model of a Transaction Node

At each of these nodes there is a TPSUI, and the dialogues that allow it to communicate with its superior and subordinate TPSUIs (if any).

Each dialogue passes through a common Multiple Association Control Function (MACF), and use a separate Single Association Control Function (SACF). The MACF is fully aware of the exchanges passing across the dialogue through each SACF.

The OSI TP protocol Machine (TPPM) provides the TP services to a single TPSUI and encompasses the TP logic and protocol for a single node in the transaction tree. The TPPM includes the MACF and SACFs, and provides the TP services to the user.

The TPSUI and/or the TPPM (on behalf of the TPSUI) may have local resources under their control that are part of the distributed data of the transaction.

The root TPPM coordinates the commitment of the nodes in the transaction tree. Each TPPM in the transaction instructs its TPSUI to coordinate its local resources (if any), and coordinates local resources on behalf of the TPSUI (if any).

The TPSUI can be informed of the start of a transaction (on arrival of a TP-BEGIN-DIALOGUE indication or by arrival of a TP-BEGIN-TRANSACTION indication on an existing dialogue), or it may start a transaction itself. The TPSUI can then extend the transaction tree to subordinates by either establishing transaction mode dialogues or commencing transaction branches on existing dialogues. All of these transaction mode dialogues share a single MACF.

TPPM service primitives are either local or global in scope. Local primitives apply to a single dialogue, for example, TP-GRANT-CONTROL request. Global primitives apply to all the coordinated dialogues of the transaction node, for example, TP-COMMIT request, TP-ROLLBACK request and TP-DONE request.

If the TPSUI has a superior, it is instructed to prepare for commitment by receipt of a TP-PREPARE indication on the dialogue from the superior. Alternatively, if the TPSUI has completed its work for a transaction it can, as a local decision, prepare its local resources without having received a TP-PREPARE indication from its superior; however, it may not issue a TP-COMMIT request until it receives a TP-PREPARE indication and it may be required to make further modifications to local resources to satisfy unexpected transaction semantics (for example, the superior may abandon the transaction and instruct the subordinate to rollback, or further requests may be received prior to the TP-PREPARE indication). In response to a TP-PREPARE indication the TPSUI indicates the success or failure of bringing its local resources (if any) into the READY state, and indicates success or failure by a single TP-COMMIT request or TP-ROLLBACK request to the MACF. A TP-COMMIT request indicates the TPSUI's willingness to participate in commitment of the transaction. The MACF issues prepare requests to the subordinates of the node in the transaction tree. When all subordinate nodes have reported READY, and the local TPSUI has indicated its willingness to commit (by issuing a TP-COMMIT request), the MACF reports READY to its superior.

When all resources in the transaction tree have reached the ready state, and the TPSUI at the ROOT node has instructed its MACF to commit (by issuing a TP-COMMIT request), three actions occur at each node:

- The TPPM sets any local resources under its control to the committed (final) state.
- The TPPM passes a single TP-COMMIT indication to the TPSUI (to instruct it to commit its local resources).
- The MACF within the TPPM propagates the commit decision to its subordinate nodes without external intervention.

The OSI TP model allows a superior TPSUI individually to prepare selected subordinate transaction branches by issuing TP-PREPARE requests on those branches rather than a TP-COMMIT request to the node, which prepares all subordinate branches. When each prepared branch becomes "ready" the TPPM indicates this to the superior TPSUI through a TP-READY indication.

Failure of a resource to reach the ready state is reported to the ROOT node by the transaction subtree that experienced the failure, where the MACF indicates the decision to rollback to the local TPSUI and propagates the decision to all its subordinate subtrees that did not experience the failure. At each READY node in the transaction tree, the MACF passes a TP-ROLLBACK indication to the local TPSUI, and propagates the rollback decision to subordinate nodes in the transaction tree.

The TPSUI indicates to the MACF the result of the commitment (or rollback) of local resources by issuing a TP-DONE request.

To sum up:

- The MACF within a TPPM is fully aware of the transaction management protocol over each dialogue.
- The MACF within the TPPM exchanges PDUs on all the coordinated dialogues for a node during commitment and rollback, without the involvement of the TPSUI in the exchanges on a individual dialogue — it coordinates the exchanges on individual dialogues (in a similar manner to the TM coordinating multiple RMs).

- The TPSUI coordinates the preparation, commitment and rollback of any local resources under its control (OSI TP regards them as a single entity).
- The TPPM coordinates the preparation, commitment and rollback of any local resources under its control.

2.1.4 Relationship between OSI TP and DTP Models

In the OSI TP model a node in the transaction tree comprises a TPSUI and a TPPM consisting of a MACF and coupled SACFs (with their dialogues). In the X/Open DTP model there is an AP, a TM and one or more RMs. The following description explains how these two views fit together.

The coordination of multiple local resources by the TPSUI is outside the scope of the OSI TP model. This coordination of local resources is performed by the TPSUI. The internal structure of a TPSUI, with its coordination of local resources, can be exemplified by the DTP model as shown in Figure 2-4.

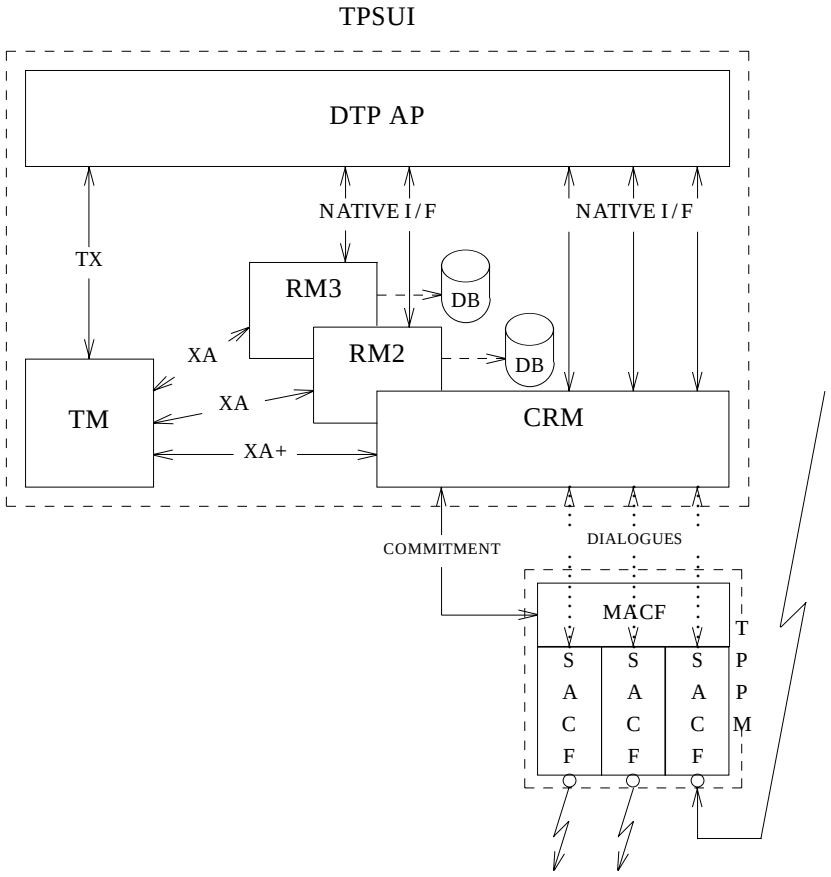


Figure 2-4 Mapping of a TPSUI to DTP Model

From a DTP model perspective, OSI TP is represented as a single resource via a CRM. The internal coordination of multiple dialogues performed by the MACF during commitment and rollback is not visible in the TPSUI. This is analogous to the database case where the database consists of multiple resources coordinated internally, but represented as a single resource by an RM. The CRM may be either superior to (when the DTP AP is not the ROOT node), or subordinate of (when the DTP application is the ROOT node), the DTP application and the TM.

From an OSI TP model perspective, the CRM, the DTP AP and the TM taken together can be viewed as a single TPSUI.

The CRM relays commitment and rollback exchanges between the MACF and the TM through the XA+ interface. Exchanges on individual dialogues are routed to the DTP AP after any local processing by the CRM.

When the DTP AP is the ROOT node of the transaction, the TM coordinates commitment of the RMs. All the RMs are subordinate to the TM. In this situation it is not possible for the CRM to issue a commit request to the TM. The OSI TP MACF still generates a TP-COMMIT indication to instruct the TPSUI to commit the local resources, and expects a TP-DONE request when this commitment is complete. As the local resources are already being committed under the control of the TM, and as the CRM is subordinate to the TM, there are no local resources for the TPSUI to commit in response to this request from the MACF. The CRM therefore immediately issues a TP-DONE request indicating successful commitment of the (zero) local resources to the MACF.

The outcome of the OSI TP commitment of subordinate nodes is reported back, via the CRM, to the TM which collates the results from all RMs and reports the overall outcome to the DTP AP as required.

The X/Open DTP model treats all subordinate RMs and CRMS in the same manner. Each is first requested to bring their resources into the ready to commit state by a call to *xa_prepare*( ) and when all have reported success they are all instructed to commit by calls to *xa_commit*( ).

The OSI TP model, however, requires that the TPSUI prepare its local resources (bringing them to the ready-to-commit state) before issuing a TP-COMMIT request. On receipt of the TP-COMMIT request, the OSI TP MACF completely controls both phases of commitment without allowing the TPSUI to intervene.

The outcome of this is that the two models cannot be integrated as they stand.

In order to integrate the two models, an XAP-TP implementation of OSI TP separates the two phases of commitment in OSI TP so that they can be explicitly driven within the X/Open DTP model.

The two phases of commit are performed in XAP-TP using the following primitives:

TP_PREPARE_ALL_REQ
: To request completion of the first phase of commitment.

TP_READY_ALL_IND
: To indicate completion of the first phase of commitment.

TP_COMMIT_REQ
: To commence the second phase of commitment.

The primitive TP_COMMIT_REQ in XAP-TP is not therefore the equivalent of the TP-COMMIT request abstract service definition in ISO/IEC 10026-2:1992 (the OSI TP Service). Rather, the combination TP_PREPARE_ALL_REQ, TP_READY_ALL_IND and TP_COMMIT_REQ is the equivalent of the TP-COMMIT request abstract service defined in ISO/IEC 10026-2:1992 (the OSI TP Service).

The separation of the 2 phases of commitment in this way requires changes to the OSI TP state table definitions. Two new states are introduced into the OSI TP service state table:

- AP_TP_PREPARING
- AP_TP_LOGGING_READY

and the state AP_TP_WCOMMITind (S20) has different semantics under XAP-TP.

When a TP_PREPARE_ALL_REQ is issued, the state is changed to AP_TP_PREPARING and the first phase of commitment is commenced with subordinates. If a failure occurs in this state, the transaction is rolled back. If all subordinates report ready, the state is changed to AP_TP_LOGGING_READY and the primitive TP_READY_ALL_IND is returned to the user. At this point, if a failure occurs it is not known if the XAP-TP user wishes to commit the transaction or rollback the transaction, so the OSI TP TPPM is maintained in its current state and any recovery attempts are refused with the response *retry-later* (it behaves as if it were a node with a dialogue to a superior in the READY state).

The XAP-TP user requests commencement of the second phase of commitment by issuing a TP_COMMIT_REQ primitive. The action taken depends on whether the node has an OSI TP superior branch or not, as follows:

- With an OSI TP superior, it changes state to AP_TP_WCOMMITind, issues a ready to the superior, and awaits the instruction to commit or rollback. If a failure occurs, the TPPM will instigate recovery to discover the outcome of the transaction from the OSI TP superior.
- Without an OSI TP superior, the state is changed to AP_TP_COMMIT_WDONEreq, a TP_COMMIT_IND primitive is issued to the user, and commit is propagated to subordinate dialogues. Due to the asynchronous nature of the XAP-TP interface, the interface state becomes AP_TP_WCOMMITind before the TP_COMMIT_IND primitive is received, and changes to AP_TP_COMMIT_WDONEreq when the primitive has been received. Again, a failure causes the TPPM to instigate recovery to determine the outcome of the subordinate affected.

2.1.5 Mapping Multiple TPSUIs to a DTP AP

Another equally valid view of the relationship between the OSI TP model and the X/Open DTP model is that only a portion of the DTP AP and a CRM are contained within a TPSUI.

It is therefore possible to have multiple TPSUIs mapped to a single DTP AP. In order for the work of the TPSUIs to be coordinated into a single transaction, they must be coordinated by that portion of the X/Open DTP model lying outside the TPSUIs. This coordination is outside the scope of the OSI TP model.

An example utilising Peer-to-Peer and RDA CRMs is shown in Figure 2-5 on page 26. Note that the figure could as easily have shown an XATMI CRM or a TxRPC CRM in conjunction with a Coordinated Systems Management CRM.

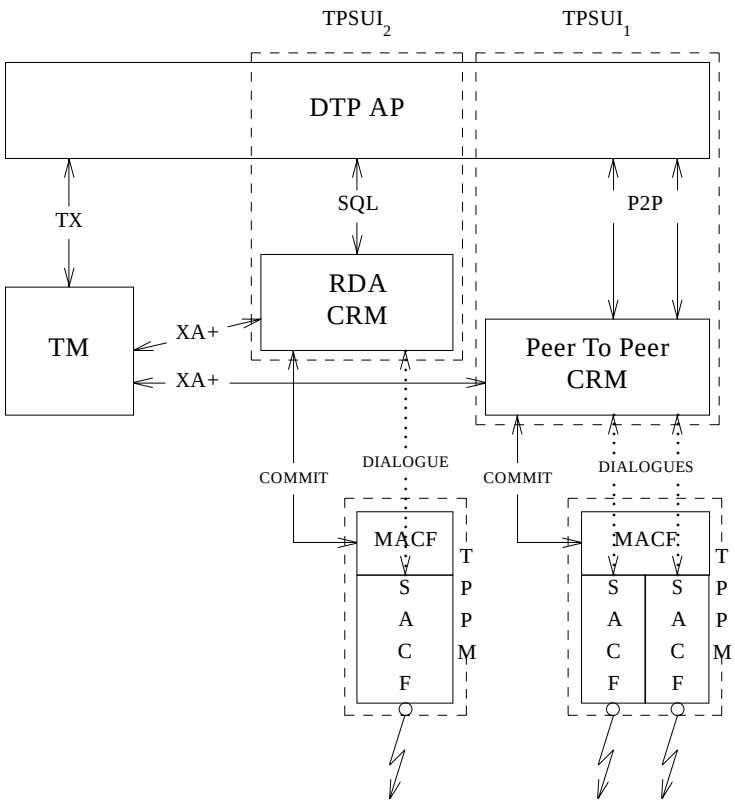


Figure 2-5 Multiple TPSUIs Mapped to a Single DTP AP

As each of the CRMs belongs to a separate TPSUI, each CRM has a separate OSI TP MACF, and they must be separately prepared and committed. The TM uses the XA+ interface of both CRMs to coordinate the overall commitment of the transaction.

As the two dialogues of the Peer-to-Peer CRM belong to the same TPSUI, they share an MACF, and the MACF coordinates the commitment flows on the dialogues. Only a single *xa_prepare*( ) and *xa_commit*( ) call is made to the CRM from the TM.

If the Peer-to-Peer CRM implementors had chosen to separate the dialogues into two separate TPSUIs, then each would have had a separate MACF. Two instances of the CRM would be open for the transaction, and the TM would have to prepare and commit each instance separately.

Note: If the TPPMs involved come from different vendors or are different software components, and both utilise the same AAID, then each must ensure that BRIDs allocated are unique.

2.1.6 Relationship of XAP-TP to OSI TP and X/Open DTP Models

XAP-TP provides a well-defined interface to the services of OSI TP MACF and SACF.

As has been seen in Section 2.1.4 on page 23, the integration of the OSI TP procedures with the X/Open DTP model requires a CRM, to:

- provide an XA+ mapping to an DTP TM
- provide a U-ASE and native interface to a DTP AP.

Applications portability for communicating X/Open DTP applications is being provided by the DTP interfaces for Client/Server (XATMI), Peer-to-Peer (P2P), and Transactional RPC (TxRPC), each being represented by a CRM. XAP-TP provides the means to construct each of these, and other RMs requiring OSI TP access, independent of the underlying OSI TP implementation. This provides system portability and component interchangeability for DTP-conformant TP systems.

Its position in respect to components of the X/Open DTP model is shown in Figure 2-6.

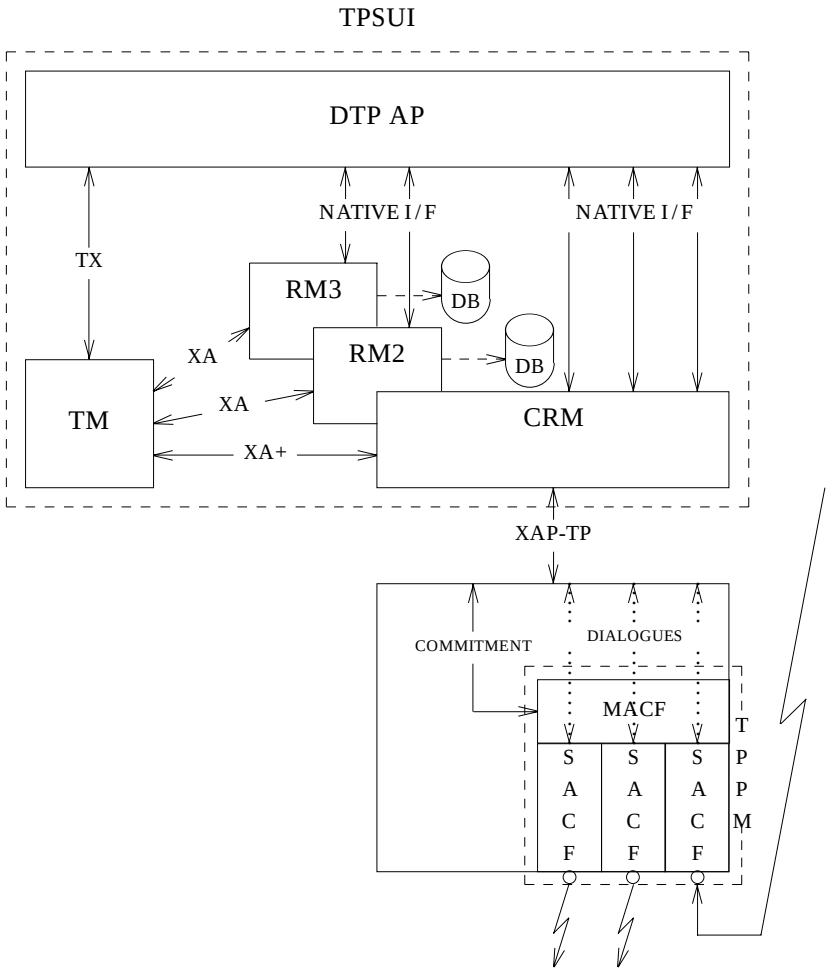


Figure 2-6 XAP-TP in Relation to the X/Open DTP and OSI TP Models

2.2 XAP-TP Functions and Mechanisms

XAP-TP uses the existing functions and mechanisms of XAP (see the **XAP** specification).

2.2.1 Selection of TP Mode

A user selects the use of the TP extension on an XAP instance by setting the AP_MODE_SEL environment attribute to AP_TP_MODE. The AP_MODE_AVAIL environment attribute has AP_TP_MODE set in an implementation supporting this specification (other modes of operation may be available as defined in the **XAP** specification and other XAP extension specifications).

A particular implementation may make OSI TP access available either:

- through a separate communications provider from ACSE / Presentation and other extensions
- through the same communications provider as ACSE / Presentation and / or other extensions.

When OSI TP access is made available through a separate provider, only AP_TP_MODE will be available in AP_MODE_AVAIL. If OSI TP access is made available through a common provider, multiple modes are available in AP_MODE_AVAIL, of which one is AP_TP_MODE.

Note: Only one mode can be selected on an XAP instance at a time with AP_MODE_SEL, when OSI TP access is made available through a common provider.

2.2.2 Categories of TP Service Primitives

The XAP-TP primitives are split into two categories:

1. dialogue primitives, which apply to an individual dialogue
2. control primitives, which are individual primitives that apply to all transaction mode dialogues of a TPSUI; the commit, logging and recovery primitives.

Note: These groupings do not directly reflect the OSI TP functional units.

The category (or categories) of TP service primitives to be used on an XAP instance are selected by bit settings of the AP_TP_CATEGORY environment attribute. The two categories can be selected singly or in combination. At least one category must be selected on an XAP instance supporting TP.

The categories are AP_TP_DIALOGUE and AP_TP_CONTROL.

An XAP instance with the AP_TP_CONTROL category selected but not supporting a dialogue can be used to send and receive commit, logging and recovery primitives for any TPSUI that is bound to the same local AET.

An XAP instance supporting a transaction mode dialogue and which has the AP_TP_CONTROL category selected can only send and receive commit and logging primitives for the TPSUI to which the dialogue belongs. Further, an XAP instance which supports a dialogue *not* in transaction mode and which has the AP_TP_CONTROL category selected *cannot* send or receive any commit or log primitives until either the dialogue ends or it is placed into transaction mode.

Later sections of this chapter explain how the various services of OSI TP are made available through an XAP instance or instances.

2.2.3 Sending and Receiving XAP-TP Service Primitives

The services offered by the OSI TP service provider are made available to the service user through a collection of XAP-TP service primitives that are sent and received using the *ap_snd*( ) and *ap_rcv*( ) functions.

The OSI TP services available through this version of the XAP-TP interface for each of the categories are shown in Table 2-1 and Table 2-2 on page 30, together with the related XAP-TP service primitives.

The service primitives available on a particular XAP instance for a dialogue are restricted to those permitted by the combination of functional units selected.

Services	Service Primitives			
	Send		Receive	
APM_ALLOCATE *	..._REQ			..._CNF
APM_ASSOCIATION_LOST *			..._IND	
A_ABORT	..._REQ			
TP_BEGIN_DIALOGUE	..._REQ	..._RSP	..._IND	..._CNF
TP_BEGIN_TRANSACTION	..._REQ		..._IND	
TP_COMMIT			..._IND	
TP_COMMIT_COMPLETE			..._IND	
TP_DATA	..._REQ		..._IND	
TP_DEFERRED_END_DIALOGUE	..._REQ		..._IND	
TP_DEFERRED_GRANT_CONTROL	..._REQ		..._IND	
TP_DIALOGUE_LOST *			..._IND	
TP_END_DIALOGUE	..._REQ	..._RSP	..._IND	..._CNF
TP_FLUSH *	..._REQ			
TP_GRANT_CONTROL	..._REQ		..._IND	
TP_HANDSHAKE	..._REQ	..._RSP	..._IND	..._CNF
TP_HANDSHAKE_AND_GRANT_CONTROL	..._REQ	..._RSP	..._IND	..._CNF
TP_HEURISTIC_REPORT			..._IND	
TP_PREPARE	..._REQ		..._IND	
TP_P_ABORT			..._IND	
TP_READY			..._IND	
TP_REQUEST_CONTROL	..._REQ		..._IND	
TP_ROLLBACK			..._IND	
TP_ROLLBACK_COMPLETE			..._IND	
TP_U_ABORT	..._REQ		..._IND	
TP_U_ERROR	..._REQ		..._IND	

* Local service not defined in ISO / IEC 10026-2: 1992 (the OSI TP Service).

Table 2-1 XAP-TP Dialogue Category Service Primitives

Services	Service Primitives			
	Send		Receive	
TP_COMMIT **	..._REQ		..._IND	
TP_COMMIT_COMPLETE			..._IND	
TP_DONE	..._REQ			
TP_DIALOGUE_LOST *			..._IND	
TP_LOG_DAMAGE *			..._IND	
TP_MANAGE *	..._REQ			
TP_NODE_STATUS *			..._IND	
TP_PREPARE_ALL *	..._REQ			
TP_READY_ALL *			..._IND	
TP_RECOVER *	..._REQ			
TP_RESTART *	..._REQ			
TP_RESTART_COMPLETE *	..._REQ		..._IND	
TP_RESUME *	..._REQ			
TP_RESUME_COMPLETE *			..._IND	
TP_ROLLBACK	..._REQ		..._IND	
TP_ROLLBACK_COMPLETE			..._IND	
TP_UPDATE_LOG_DAMAGE *	..._REQ			

* Local services not defined in ISO / IEC 10026-2: 1992 (the OSI TP Service).

** The TP_COMMIT_REQ primitive in XAP-TP is not the equivalent of the TP_COMMIT request service definition in ISO / IEC 10026-2: 1992 (the OSI TP Service). Rather, the combination of TP_PREPARE_ALL_REQ, TP_READY_ALL_IND and TP_COMMIT_REQ primitives is the equivalent of the TP_COMMIT request abstract service defined in ISO / IEC 10026-2: 1992 (the OSI TP Service).

Table 2-2 XAP-TP Control Category Service Primitives

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

Services that can be initiated by the service user may be associated with either one, two or four service primitives, depending on whether or not the service is confirmed. Figure 2-7 illustrates how an initiator and a responder may use the *ap_snd*( ) and *ap_rcv*( ) functions, together with the TP_HANDSHAKE service primitives, to synchronise their processing.

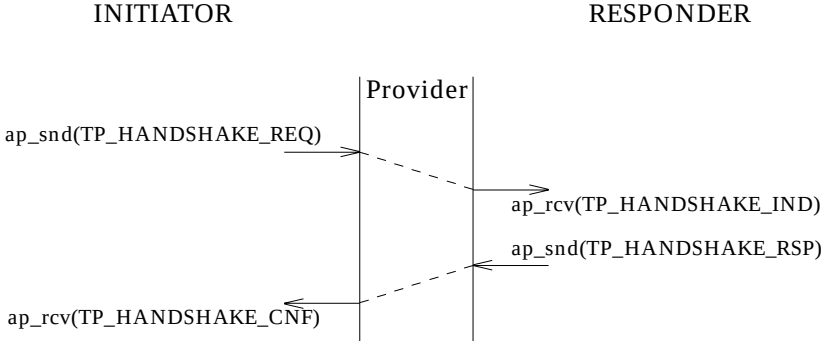


Figure 2-7 Synchronising a Dialogue using Handshake

Note that when using the XAP-TP library with a provider which supports both AP_TP_MODE and AP_NORMAL_MODE, only one mode can be selected on an individual XAP instance at a

time. An individual XAP instance may be used to drive an association using the primitives from the **XAP** specification or, alternatively, to drive a dialogue using the primitives from this specification.

2.3 OSI TP Address Lookup and Directories

The XAP-TP interface uses the addressing from the OSI TP Service Standard. This is at the level of AETs. The AETs in use for branches of a transaction are amongst the information stored in the log record for the TPSUI to enable recovery to be effected.

The use of the AETs during recovery when the application may no longer be available precludes the application from providing the lower-level addresses that these AETs map to. The need to enable relocation of failed systems to alternative hardware, with its consequent change of addressing, precludes the addresses being stored in the log records.

The means used to convert an AET into the addressing information used by presentation and below is implementation-dependent. Typically an XAP-TP implementation would use a local configuration file or a directory search to find the address to which an AET maps.

For recovery purposes, it may be necessary to relocate an AET to different hardware (at a different address) so a local mechanism should be provided to allow the address to which an AET maps to be changed in the running system.

2.4 Association Allocation and Deallocation

For outgoing TP_BEGIN_DIALOGUE_REQ primitives an association must be allocated. This can be performed in one of two ways:

Automatically
: by ensuring the AP_TP_ASSOC_ALLOCATED bit is UNSET in the *cdata→tp_options* field on the send of a TP_BEGIN_DIALOGUE_REQ primitive

Specifically
: by using the APM_ALLOCATE_REQ primitive to allocate an association prior to issuing a TP_BEGIN_DIALOGUE_REQ primitive with the AP_TP_ASSOC_ALLOCATED bit set in the *cdata→tp_options* field.

In automatic allocation, the user issues a TP_BEGIN_DIALOGUE_REQ primitive, which is internally queued within the XAP-TP provider whilst it performs association allocation. An APM_ALLOCATE_CNF primitive is issued to the user to report the success or failure of association allocation and TP_BEGIN_DIALOGUE_REQ submission. If success is reported, the queued TP-BEGIN-DIALOGUE request has been submitted, otherwise it has been silently discarded and no association has been allocated.

In specific allocation, the user issues an APM_ALLOCATE_REQ primitive to perform association allocation. Once again, an APM_ALLOCATE_CNF primitive is issued to the user to report the success or failure of association allocation.

When specific allocation is employed, the user may receive an APM_ASSOCIATION_LOST_IND primitive if the association becomes unavailable prior to the user issuing the TP_BEGIN_DIALOGUE_REQ. This can occur because either:

- The association has ceased to exist.
- An incoming TP_BEGIN_DIALOGUE_IND has arrived on the association, causing it to be allocated to another XAP-TP instance.

It is an error to attempt combining both forms of association allocation. Specifically attempts to:

- issue a TP_BEGIN_DIALOGUE_REQ without the AP_TP_ASSOC_ALLOCATED bit set in *cdata→tp_options* when an association has been allocated with APM_ALLOCATE_REQ
- issue a TP_BEGIN_DIALOGUE_REQ with the AP_TP_ASSOC_ALLOCATED bit set in *cdata→tp_options* when no association is currently allocated to the XAP-TP instance

cause an error code to be returned.

With specific or automatic allocation, the association will be deallocated when one of the following conditions occurs:

- The dialogue supported on it ends.
- The association is lost.
- The XAP-TP instance is closed without a TP_BEGIN_DIALOGUE_REQ primitive having been issued.

At this point, it is returned to an association pool for re-use, or freed, based on a local implementation decision.

2.5 Mapping TPSUIs to Processes

The OSI TP specification does not constrain the mapping of TPSUIs to real processes on a real end system. The XAP-TP specification allows the user flexibility in the choice of this mapping:

- many TPSUIs in one process
- single TPSUI spread across many processes (for example, one per dialogue).

An XAP instance may either support a single OSI TP dialogue (and optionally logging and commit primitives for its TPSUI) or logging/commit primitives for the local AET to which the instance is bound.

The X/Open DTP model allows commitment and logging to be performed in a different process from the communication on dialogues. XAP-TP allows the separation of commit and logging functions for a TPSUI to one or more processes by using the AP_TP_CATEGORY environment attribute to select the AP_TP_CONTROL category of primitives on an XAP instance.

On those implementations that support the functions, *ap_save*( ) and *ap_restore*( ) may be used to drive a dialogue in multiple processes.

2.6 Control of Dialogue Tree Structure

As a single node of a dialogue tree may be distributed over many processes, and multiple nodes from different dialogue trees may be contained within a single process, it is necessary to identify to which node a particular dialogue belongs when establishing it.

This allows the dialogues of a node to share an OSI TP MACF, and so to be committed and logged as a unit in transaction mode.

When a dialogue arrives from a superior, XAP-TP allocates a unique Dialogue Tree Node Identifier (DTNId) and places this in the AP_DTNID environment attribute.

A user can retrieve the AP_DTNID attribute and set it on another XAP instance prior to initiating a dialogue to tie the new dialogue to the same node. This is shown in Figure 2-8.

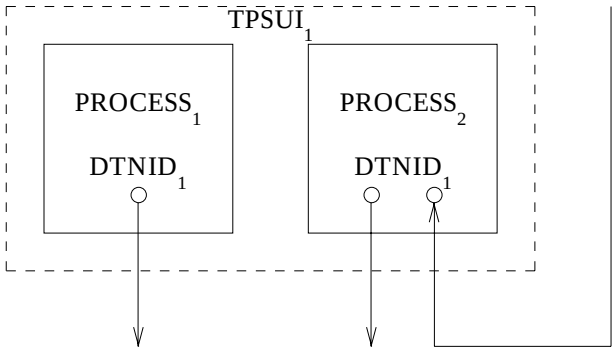


Figure 2-8 One TPSUI Across Several Processes

If the user does not set the AP_DTNID environment attribute (or if it is set to NULL), then XAP-TP will allocate a new DTNId and the dialogue will belong to a newly created root node in a separate dialogue tree when a TP_BEGIN_DIALOGUE_REQ is issued. If this dialogue is subsequently placed into transaction mode, it will be separately logged and committed.

An application which requires control over the phases of commitment for each dialogue can ensure this by setting AP_DTNID to NULL prior to starting new dialogues. Figure 2-9 shows the effect of this.

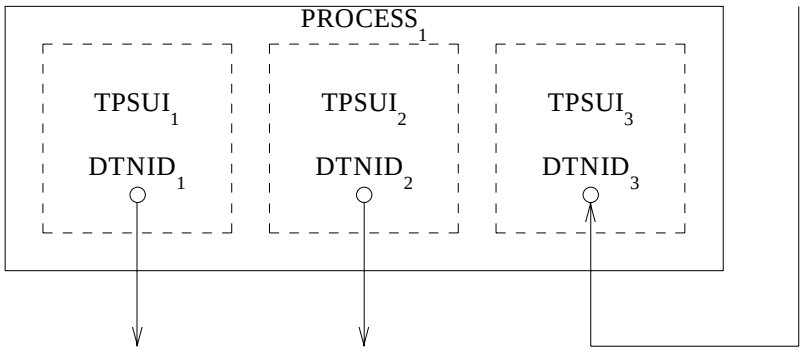


Figure 2-9 Several TPSUIs within a Process

Constraints on the Dialogues of a Node in the Dialogue Tree

All XAP-TP instances supporting dialogues of the same node in the dialogue tree are constrained to have the same values for the following environment attributes:

- AP_LCL_APT
- AP_LCL_AEQ
- AP_LCL_APID
- AP_LCL_AEID
- AP_LCL_TPSUT

All dialogues of the node with the commit functional unit selected are supported on associations having the same value of user-assigned recovery context handle (see Section 2.11 on page 44).

Uniqueness of Dialogue Tree Node Identifier Values

Dialogue tree node identifier values are only required to be unique within the scope of a local AET.

If a single software product is using multiple local AETs, it should take into account that the dialogue tree node identifier values returned from instances bound to different local AETs may be the same.

2.7 Control of the Transaction Tree Structure

As we have seen in the OSI TP overview, the branches of a transaction tree are supported on the branches of a dialogue tree, and all the branches of the transaction tree for a node (TPSUI + TPPM) share a single OSI TP MACF. They are therefore logged and committed as a whole.

Using the AP_DTNID attribute, the user can control the construction of the dialogue tree. The normal TP_BEGIN_DIALOGUE (in transaction mode) and TP_BEGIN_TRANSACTION primitives allow the user to control which dialogues of the tree are also in the transaction tree.

Using a Local Identifier

In a TP system there often exists a local identifier which is more convenient for referring to the transaction on a particular node. XAP-TP allows this identifier to be associated with the node and it may be used as an alternative to the DTNId when:

- adding further branches to the dialogue tree
- issuing and receiving commit or log category primitives for the transaction node.

The identifier is called the Transaction Tree Node Identifier, and is held in the AP_TTNID environment attribute. The user may set the AP_TTNID environment attribute from any of the XAP instances supporting a dialogue for the TPSUI. Note that there only exists a single AP_TTNID (and AP_DTNID) for all of the dialogues of a node — changing the AP_TTNID on one XAP instance affects all others.

The AP_TTNID is secured in a nodes log record by XAP-TP to protect it from system crashes during commitment. It is the user's responsibility to set the attributes AP_NEXT_TTNID and AP_TTNID to a new value as needed.

A superior node must set the AP_TTNID attribute before issuing a TP_BEGIN_DIALOGUE_REQ or TP_BEGIN_TRANSACTION_REQ primitive in order to start a transaction branch, and a subordinate node must set the attribute after receiving a TP_BEGIN_DIALOGUE_IND or TP_BEGIN_TRANSACTION_IND primitive which starts a transaction branch.

When a transaction node is chaining, the AP_NEXT_TTNID attribute must be set:

- before issuing a TP_COMMIT_REQ primitive for the current transaction
- before issuing a TP_DONE_REQ primitive for the current transaction when rollback has been initiated.

Before indicating termination of the current transaction by issuing a TP_COMMIT_COMPLETE_IND or TP_ROLLBACK_COMPLETE_IND primitive, the provider copies the value of AP_NEXT_TTNID into the AP_TTNID attribute and unsets the AP_NEXT_TTNID attribute.

When a transaction node is not chaining, the AP_NEXT_TTNID attribute is not used. On transaction termination, the AP_TTNID attribute is unset and must be set to a new value prior to initiating another transaction, or after receiving indication of the start of another transaction, as described above.

Uniqueness of Transaction Tree Node Identifier Values

Transaction tree node identifier values are only required to be unique within the scope of the individual recovery context group they are being used in. If separate explicit recovery context groups are not being used for an AET, transaction tree node identifier values must be unique within the scope of the AET.

If multiple software products are cooperating in using the XAP-TP interface, they must cooperate to ensure that the transaction tree node identifier values remain unique.

2.8 User Setting of AAID and BRID

Where permitted by the OSI TP protocol, the user may choose to allocate *Atomic Action Identifiers* (AAIDs) and/or *Branch Identifiers* (BRIDs), or alternately allow the provider to allocate them.

At a ROOT node of the transaction tree, the global attribute AP_AAID holds the current *Atomic Action Identifier* in use on the transaction tree node, or the *Atomic Action Identifier* to be used for the next transaction on the node if no current transaction exists. If the user leaves AP_AAID unset (or specifically removes it from the environment by issuing an *ap_set_env*( ) call with a NULL pointer as the val argument), the OSI TP provider will generate a new AAID for the transaction when it starts and place its value into the AP_AAID attribute.

Similarly, the user may set the AP_BRID attribute for a dialogue prior to commencing a transaction branch. The value from AP_BRID is used for the *Branch Identifier* of the transaction branch. If the user leaves AP_BRID unset (or specifically removes it from the environment by issuing an *ap_set_env*( ) call with a NULL pointer as the val argument), the OSI TP provider will generate a new BRID for the transaction branch and place its value into the AP_BRID attribute.

Note that when the user provides *Atomic Action Identifiers* and/or *Branch Identifiers*, they must conform to the requirements of the OSI TP protocol, specifically the *Branch Identifiers* must contain the AET of the node.

At an INTERMEDIATE or LEAF node in the transaction tree, the *Atomic Action Identifier* is received from the superior and is placed in the AP_AAID attribute by the XAP-TP implementation. Similarly, the *Branch Identifier* is received from the superior and is placed into the AP_BRID attribute of the XAP-TP instance supporting the superior dialogue. The user may choose to specifically set the *Branch Identifier* values for subordinate transaction branches or to allow the OSI TP provider to allocate them as described above.

When a transaction node is not chaining and a transaction terminates, the AP_AAID attribute is unset. The AP_BRID attribute remains unchanged.

Transaction Chaining

When chained transactions are in use, the attributes AP_NEXT_AAID and AP_NEXT_BRID allow the user to specify the *Atomic Action Identifier* and *Branch Identifier* to be used for the next transaction in a chain. The value present in AP_NEXT_AAID (and AP_NEXT_BRID) is used at different times by the OSI TP implementation depending on whether rollback or commitment occurs.

When rollback is initiated, the user may set a new value for a *Branch Identifier* into the AP_NEXT_BRID attribute of all the chaining subordinate dialogues of the node prior to issuing the first TP_DONE_REQ primitive for the transaction node rolling back. At rollback completion, for a chaining subordinate dialogue, the value from AP_NEXT_BRID becomes the new *Branch Identifier* for the branch, is copied into the AP_BRID attribute, and AP_NEXT_BRID is set NULL by the provider.

The user may place a new *Atomic Action Identifier* value into the AP_NEXT_AAID attribute prior to issuing the first TP_DONE_REQ primitive for the transaction node. Note that the value in AP_NEXT_AAID is only used if the node is the root of the transaction (or becomes the root of the transaction tree) at rollback completion.

Similarly, for commitment, any values for AP_NEXT_AAID and AP_NEXT_BRID must be set prior to issuing the TP_COMMIT_REQ primitive for the transaction node.

If the node is the root of a new transaction at transaction completion, the AP_NEXT_AAID value becomes the *Atomic Action Identifier* for the node, is copied into the AP_AAID attribute, and AP_NEXT_AAID is set NULL by the provider. If the node is not a ROOT node, a new *Atomic*

Action Identifier is received from the superior and is placed into the AP_AAID attribute. Any value in the AP_NEXT_AAID attribute remains unchanged.

During commit completion, for a chaining subordinate dialogue, the value from AP_NEXT_BRID becomes the new *Branch Identifier* for the branch. It is copied into the AP_BRID attribute and AP_NEXT_BRID is set NULL by the provider.

If the user leaves AP_NEXT_BRID unset (or specifically removes it from the environment by issuing an *ap_set_env*( ) call with a NULL pointer as the val argument) the existing *Branch Identifier* will be reused for the new transaction branch. If the user leaves the AP_NEXT_AAID attribute unset (or specifically removes it from the environment by issuing an *ap_set_env*( ) call with a NULL pointer as the val argument), and the transaction node commences another transaction as a ROOT node, the OSI TP provider will generate a new *Atomic Action Identifier* and place its value into the AP_AAID attribute.

Note that if the user application, with a chaining superior dialogue, wishes to exercise control over the allocation of AAIDs and can tolerate becoming the ROOT of the transaction tree, it should set the AP_NEXT_AAID attribute for this eventuality prior to issuing the first TP_DONE_REQ primitive during rollback or before issuing a TP_COMMIT_REQ primitive prior to commitment.

2.9 U-ASE Support in XAP-TP

2.9.1 Types of U-ASE

U-ASEs can be divided into 2 types: simple and complex U-ASEs.

Complex U-ASEs do one or more of the following:

- exchange pdus at association establishment time
- maintain state on the association independently of the presence or absence of a dialogue
- directly access association facilities (for example, minor sync)
- change the default mapping of TP pdus.

Examples are RDA and CMISE.

Simple U-ASEs:

- do not exchange pdus at association establishment time
- transfer data on an association only within the bounds of a dialogue
- do not maintain state on the association independent of the presence or absence of a dialogue
- do not directly access association facilities (for example, minor sync)
- do not change the default mappings of TP pdus.

Examples are XATMI-ASE and TxRPC RPC-ASE.

Complex U-ASEs are embedded below the XAP-TP interface as shown in the diagrams in ISO / IEC 10026-3: 1992 (the OSI TP Protocol). Simple U-ASEs may be either embedded below or reside above the XAP-TP interface. XAP-TP supports U-ASEs both above and below the interface through the TP_DATA_REQ and TP_DATA_IND primitives.

For a particular application context, the U-ASEs must be either all above or all below the XAP-TP interface.

Local configuration information (outside the scope of this specification) defines for a particular application context whether its component U-ASEs reside above or below the XAP-TP interface.

2.9.2 U-ASEs Below the XAP-TP Interface

The XAP-TP user utilises the TP_DATA_REQ primitive with the *ap_snd*( ) function to pass a request (or fragment of a request) to a particular U-ASE of the application context. Each fragment of a request except the last has the AP_MORE flag set on the *ap_snd*( ) function call.

An embedded U-ASE passes an indication (or fragment of an indication) to the user through a TP_DATA_IND primitive.

The *user_id* field of the *cdata* structure holds the index number, within the application context, of the U-ASE to which a request or indication belongs (indices commence at 0).

The data formats of the U-ASE service primitives passed in the data buffers of TP_DATA_REQ and TP_DATA_IND primitives are defined in the U-ASE specifications.

2.9.3 U-ASEs Above the XAP-TP Interface

A U-ASE above the XAP-TP performs its own pdu encoding and decoding, submitting each U-ASE PDV-List to XAP-TP as a separate TP_DATA_REQ.

All TP_DATA_REQ primitives are passed to a simple U-ASE support function component which submits each PDV-List for concatenation according to the TP and CCR concatenation rules. If there are no TP or CCR pdus buffered, a new concatenation sequence is commenced and will be carried on P-DATA unless subsequent TP or CCR PDUs require a different carrier, in which case the required TP or CCR carrier takes precedence.

A U-ASE can use the TP_FLUSH primitive or AP_FLUSH flag on *ap_snd*( ) to end a concatenation sequence.

When a concatenation sequence is received and separated, the simple U-ASE support component passes each U-ASE PDV-List to the XAP-TP user as a separate TP_DATA_IND primitive.

2.10 Explicit Control of the Two Phases of Commit

Coordination with Local Resources

Provision of TP_PREPARE_ALL_REQ and TP_READY_ALL_IND primitives allows the first phase of commitment to be performed in parallel with preparation of local resources.

Coordination with an External MACF

Note that there may be a Control Function outside the scope of the OSI TP MACF which coordinates the commitment of multiple OSI TP MACFs and other resources. XAP-TP supports this by provision of the TP_PREPARE_ALL_REQ and TP_READY_ALL_IND primitives.

2.11 Recovery Context Groups

The OSI TP recovery context handle concept may be used by an implementation to segment the set of transaction nodes into separate groups such that logging and recovery for one group can be handled independently of logging and recovery for another.

XAP-TP supports this by allowing nodes to be separated into different recovery context groups. A Recovery Context Group is identified by a user-assigned portion of the OSI TP recovery context handle.

The XAP-TP provider may also use the OSI TP Recovery Context Handle concept for its own purposes. It combines any user-assigned portion with any provider-generated parts to form the actual OSI TP Recovery Context Handle used. The user-assigned portion is held in the AP_URCH attribute, and may be from 0 to 32 octets in length.

Each different recovery context group within the scope of an AET is identified by a different AP_URCH value. The XAP-TP provider ensures that OSI TP recovery context handles created from different AP_URCH values remain unambiguous. Each Local AET is treated separately for recovery purposes, so recovery context groups having the same AP_URCH attribute, but different local AETs, are treated as separate and distinct from each other.

Dialogues of a node which have the commit functional unit selected must all have either the same recovery context handle, or no recovery context handle. This is achieved by ensuring the AP_URCH environment variable is set to the same value on each instance prior to allocating an association and establishing a dialogue.

The Default Recovery Context Group

If the user does not wish to use the recovery context group facility, the AP_URCH attribute may be left unset. As a result, either no recovery context handle will be used, or the handle used will be entirely XAP-TP provider-generated.

Selecting a Recovery Context Group

A Recovery Context Group is selected by setting the AP_URCH attribute prior to calling *ap_bind*( ) for an XAP-TP instance. An XAP-TP instance in the AP_TP_IDLE state can be moved to another recovery context group by setting a different value in the AP_URCH environment attribute and calling *ap_bind*( ) again.

Control Instances within a Recovery Context Group

Commitment and logging for transaction nodes within a recovery context group is performed through one or more XAP-TP instances with the AP_TP_CONTROL category selected. Each of these control instances are bound to the LAET and user-assigned recovery context handle of the group.

Identifying Control Instances within a Group

Each control instance in a group must be uniquely identifiable to ensure correct operation during recovery. Thus, each control instance has a group-wide identifier assigned by the user. This identifier is set in the AP_CONTROL_ID environment attribute prior to calling *ap_bind*( ) to bind a control instance to the recovery context group. If a control instance with the control identifier already exists within the recovery context group, the *ap_bind*( ) call will fail and the error [AP_TP_BAD_CONTROL_ID] will be returned.

Note: The AP_CONTROL_ID attribute has no default and must be set to a non-null value.

Nominating a Control Instance for a Node

Before a transaction can commence on a dialogue, the dialogue tree node must have a nominated control instance to route commit, rollback and log primitives to. The user identifies the control instance using its control identifier.

In order to ensure that no transaction starts on a node without a nominated control instance, XAP-TP requires that the control instance be nominated prior to allocating an association for an outgoing dialogue, and prior to commencing listening for incoming dialogues. This is therefore done as part of binding.

The user sets the AP_CONTROL_ID attribute to the control identifier of the nominated control instance prior to calling *ap_bind*( ) for the instance which is to support the first dialogue on the node.

If the nominated control instance is not available (presumably due to a failure in some other part of the system) the bind will be successful, but commitment, rollback and logging will stall waiting for XAP-TP control instance resumption (see Section 2.12.1 on page 47) until the control instance becomes available or the user changes the control instance for the node.

Changing the Control Instance for a Node

The user may change the control instance for a node by issuing a TP_MANAGE_REQ primitive for the node identified by DTNId or TTNId on the new control instance where logging, commitment or rollback is to occur. This new control instance must be within the same recovery context group as the original. A TP_NODE_STATUS_IND primitive will be issued by XAP-TP for the node on the new control instance. This enables the user to synchronise its view of the transaction state and log record contents with XAP-TP.

Due to the asynchronous nature of the XAP-TP interface, changing a control instance for a node in this manner, while the original control instance is still available, may result in primitives being received on the original instance that were issued by XAP-TP prior to the change taking effect. These redundant primitives can be safely discarded.

Persistence of Recovery Context Groups

A Recovery Context Group comes into existence within an XAP-TP provider the first time one of the following occurs:

- an XAP-TP instance with the AP_TP_CONTROL category selected is bound to the group
- an association pool is defined using the Recovery Context Handle identifying the group.

A Recovery Context Group ceases to exist within an XAP-TP provider when all of the following conditions are true:

- There are no association pools defined within the provider using the Recovery Context Handle which identifies the group.
- There are no transaction nodes present in the group (whether connected to an XAP-TP user or not — note this includes heuristic report nodes for transactions which have completed).
- There are no XAP-TP instances bound to the Recovery Context Group.

2.12 Recovery in XAP-TP

The recovery facilities of XAP-TP allow the user and XAP-TP to synchronise their processing on behalf of a transaction node after one or more of the following:

- system failure
- process failure
- XAP-TP provider failure.

From the perspective of XAP-TP, these three failures types have different effects as follows:

System Failure
: All running activity has been lost.

  The user will have to use its log to reconstruct its list of transactions and recover them. The XAP-TP provider will have lost all knowledge of transactions that were running at the point of failure. The user will have to *restart* XAP-TP to reconstruct its internal tables and recover transaction nodes in commitment at the point of failure.

Process Failure
: The effects depend on the process' involvement in XAP-TP usage, as follows:

  — None.

    If, as a result of this failure, the user discovers that its tables are in an inconsistent state, it may decide to reconstruct them and resynchronise with XAP-TP. This would be manifested to XAP-TP by the user closing the control instances affected, later re-establishing them and performing an XAP-TP *resumption* for them.

  — One or more XAP-TP instances are in use within the process.

    Each of these instances will be implicitly closed.

    If the instance supports an active dialogue, this causes an implicit TP_U_ABORT_REQ to be issued on the dialogue. If the aborted dialogue supported a transaction branch and the transaction node was not in commitment, this causes the node to rollback.

    If the instance has the AP_TP_CONTROL category selected, the control identifier will be marked unavailable within the XAP-TP provider until recovery takes place.

XAP-TP Provider Loss
: This results in all XAP-TP instances becoming unusable for one, many or all recovery context groups. All affected instances are marked unusable, and return the error [AP_TP_NO_PROVIDER] on any function call. Any incomplete call will abnormally terminate. For example, if the user is blocked in a synchronous *ap_snd*( ) call, this will return with the error value [AP_TP_NO_PROVIDER]. Each of the instances will be returned to the AP_TP_UNBOUND state and attempts to bind instances specifying the recovery context group without the AP_TP_CONTROL category selected will fail with the error code [AP_TP_NO_PROVIDER], until the affected part of the XAP-TP provider becomes operational again and the recovery context group has been successfully restarted.

Two types of recovery are provided by XAP-TP: resumption and restart.

Resumption
: XAP-TP passes the current state of each of the transaction nodes for a control instance to the XAP-TP user to allow the user to synchronise its view of the transaction state and log with XAP-TPs. It applies to the single control instance within a recovery context group as identified by its control identifier AP_CONTROL_ID.

Note: XAP-TP does not initiate rollback of all active nodes associated with a control instance when the control instance is closed. The user should therefore initiate rollback for active nodes which are unknown to it.

Restart

The user passes all the logged ready, commit and heuristic_damage records of the recovery context group to XAP-TP for it to reconstruct its internal tables and perform recovery of the transaction nodes involved. This would typically be performed after system reload or XAP-TP provider loss.

Figure 2-10 shows the primitive and state transitions for resumption and restart, and how they interact. The sections following explain the details of resume and restart in detail.

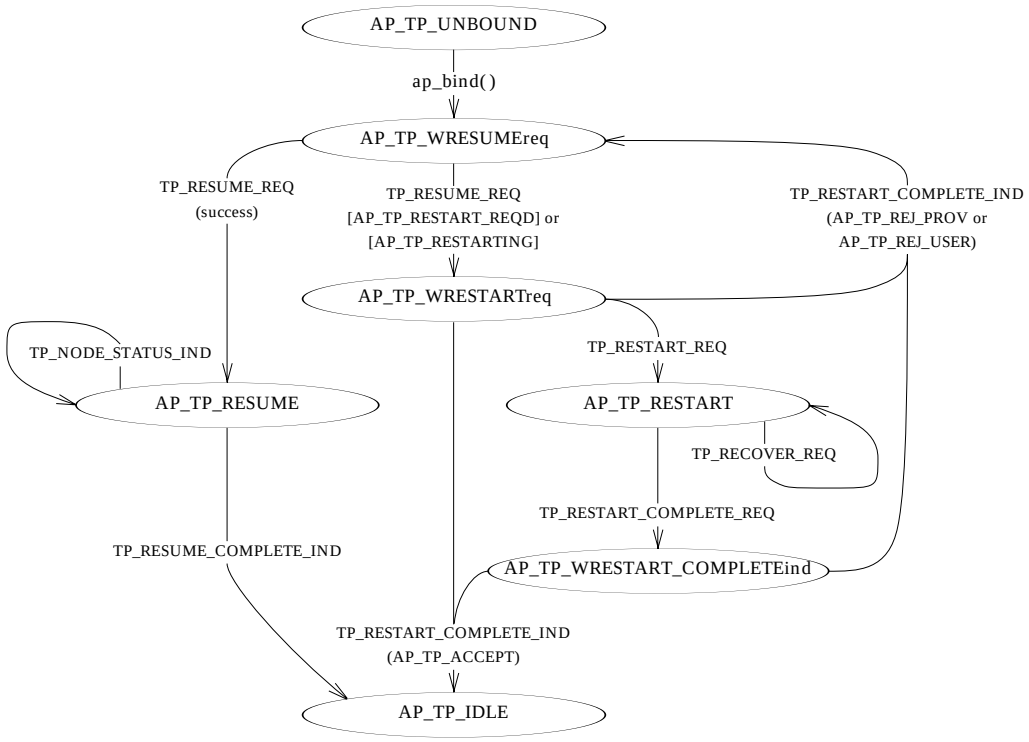


Figure 2-10 Resumption and Restart State / Event Flows

2.12.1 XAP-TP Control Instance Resumption

For an XAP-TP control instance, return from *ap_bind*( ) leaves the instance in state AP_TP_WRESUMEreq. The only valid primitive in this state is TP_RESUME_REQ.

If the recovery context group requires restarting, the TP_RESUME_REQ fails returning the error code [AP_TP_RESTART_REQD]. If restart of the group is in progress, the call fails returning the error code [AP_TP_RESTARTING]. In both cases the state of the instance changes to AP_TP_WRESTARTreq and the user must either actively or passively participate in restart of the group (see Section 2.12.2).

If the issuing of the TP_RESUME_REQ primitive succeeds, the resumption continues to return a succession of TP_NODE_STATUS_IND primitives (one for each node that has the instance as its nominated control instance) until the XAP-TP provider signals the end of resumption by returning a TP_RESUME_COMPLETE_IND primitive to indicate the resumption is complete.

The user performs the following steps to resume use of a control instance within a recovery context group:

1. issues a TP_RESUME_REQ primitive on the instance.
2. then calls *ap_rcv*( ) repeatedly to retrieve TP_NODE_STATUS_IND primitives until a TP_RESUME_COMPLETE_IND primitive is received. The AP_NODE_STATUS_IND primitive returns the following information for the node:
   — the state of the first coordinated dialogue of the node
   — its TTNId (if set)
   — its DTNId
   — the current log record for the node (if one has been issued by XAP-TP).

The dialogue state reflects the commitment state of the node and is the same value that would be returned by an *ap_get_env*( ) call to retrieve AP_STATE on the instance supporting the dialogue.

If the dialogue is in one of the active states prior to commitment or rollback and the user has no record of the transactions TTNId or DTNId, it should initiate rollback for the node.

2.12.2 XAP-TP Restart of a Recovery Context Group

Restart of a recovery context group is necessary if all knowledge of the group, and therefore the transactions within the group, has been lost. This may occur because of any of the following conditions:

- The system has become unavailable and been reloaded.
- Part or all of the XAP-TP provider has become unavailable and has been reloaded.
- Simply because the group is not currently in use.

The user is informed that a restart is required or is in progress by the return of the error codes [AP_TP_RESTART_REQD] and [AP_TP_RESTARTING] respectively from an *ap_snd*( ) of a TP_RESUME_REQ primitive. The control instance is then in state AP_TP_WRESTARTreq and the user can either participate actively in restart by issuing a TP_RESTART_REQ primitive, or participate passively by waiting for the TP_RESTART_COMPLETE_IND primitive to be received, indicating that the restart is complete.

An active participant in restart typically extracts the XAP-TP portion of log data from its internal tables or its log and passes this to XAP in a TP_RECOVER_REQ primitive. The XAP-TP provider constructs either an OSI TP TPPM or an internal heuristic_damage entry for each log record passed, as follows:

- for a log-ready record without a superior branch section:
  creates a TPPM, and places the node into state AP_TP_LOGGING_READY.
- for a log-ready record with a superior branch section:
  creates a TPPM, and places the node into state AP_TP_WCOMMITind.
- for a log-commit record:
  creates a TPPM and places the node into state AP_TP_WCOMMITind.
- for a log-damage record:
  creates an internal heuristic damage entry for the node and places the node into state AP_TP_HEURISTIC_LOG.

The user can select which control instance is to be used for further log and commit exchanges by issuing the TP_RECOVER_REQ on the control instance to be used. This may be any control

instance within the group that is actively participating in restart. In order for a control instance to actively participate in restart, a TP_RESTART_REQ must have been issued on it.

When the user has completed passing log records to the XAP-TP provider on a control instance, a TP_RESTART_COMPLETE_REQ must be issued. When a TP_RESTART_COMPLETE_REQ primitive has been received by the provider on each control instance actively participating in restart, it completes the restart by issuing a TP_RESTART_COMPLETE_IND primitive on all control endpoints which are actively or passively participating in restart, and commences operation of the recovery context group. Once operation of the group starts, outgoing recovery attempts commence and incoming recovery attempts are no longer refused (*retry-later*).

The user performs the following steps to perform restart of a recovery context group:

1. Establish one or more control instances for the group.
2. Bind each to the group using *ap_bind*( ).
3. Issue a TP_RESUME_REQ primitive on each instance — this fails, returning the error code [AP_TP_RESTART_REQD] or [AP_TP_RESTARTING].
4. Issue a TP_RESTART_REQ on each control instance to actively participate in restart.
5. For each log record in turn, the user selects the control instance for further commit, rollback and logging primitive exchanges, and issues a TP_RECOVER_REQ primitive on it passing the log record.

   Note: This must be a control instance actively participating in restart.
6. When the last log record has been passed on each active control instance, the user issues a TP_RESTART_COMPLETE_REQ primitive on each of the active control instances.
7. The user waits for a TP_RESTART_COMPLETE_IND primitive on each active and passive control instance on the group. After receipt of the primitive, the control instance is available for use.

2.12.3 Failure to Complete a Restart

If a failure occurs during the restart process such that all the control instances for the group become closed without XAP-TP having received a TP_RESTART_COMPLETE_REQ primitive, then the completeness of the information provided cannot be relied upon. XAP-TP will delete all the reconstructed TPPMs and heuristic damage nodes, and remove knowledge of the recovery context group.

Incoming recovery requests for the group will continue to be deferred with retry later responses until a restart is successfully completed for the group.

2.12.4 Unavailability of Log Records for a Recovery Context Group

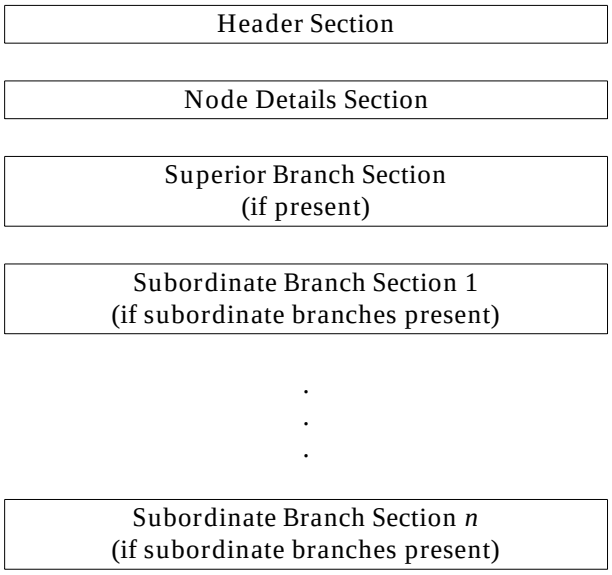
If the users log for a particular recovery context group is unavailable, for whatever reason, the user must defer the restart until the log becomes available. As no restart has been performed, XAP-TP will reject all incoming recovery attempts for the recovery context group with a response of retry later.

Other recovery context groups whose records are available can, of course, be restarted and commence normal operation.

2.13 XAP-TP Log Record Format

The XAP-TP log record is of variable length. A single log record contains all the OSI TP log record forms.

The record format consists of a number of variable-length sections.



Each section contains a number of fields. Each field may be either an integer field or a variable field. An integer field is unsigned numeric. A variable field consists of two parts: the length item and the data item. The length item is simply an integer field which gives the length of the following variable data.

When a field in the header section indicates a section is not present, then none of the fields of the section are present in the variable part.

In the following section, the notation:

```
<integer> name
```

indicates the presence of a field called *name* of type `integer`, and the notation:

```
<variable> name2
```

indicates the presence of a variable-length field called *name2* which consists of an `<integer>` length field followed by the actual data.

No padding is placed between fields or sections.

The Header Section

The header section contains basic information about the record and allows the user to determine the number and type of the optional sections present in the record.

```
/*
 * header section -- always present
 */

<integer> version    /* 1 for this version */
<integer> flags      /* bit flags -- meanings are: */
                     /* 1 -- superior's section present */
<integer> SnD_type   /* Node state. One of: */
                     /* - AP_TP_NONE */
                     /* - AP_TP_READY_LOG */
                     /* - AP_TP_COMMIT_LOG */
<integer> SldD_type  /* Log damage state. One of: */
                     /* - AP_TP_NONE */
                     /* - AP_TP_HEUR_MIX */
                     /* - AP_TP_HEUR_HAZ */
<integer> sub_count  /* count of subordinate branches */
                     /* zero if none present */
```

The following hash defines are used:

```
/* log record flags bit settings */
#define AP_TP_SUPERIOR_SECTION      1

/* log record type field values */
#define AP_TP_NONE                  0
#define AP_TP_READY_LOG             1
#define AP_TP_COMMIT_LOG            2
#define AP_TP_HEUR_MIX              1
#define AP_TP_HEUR_HAZ              2
```

The fields are used as follows:

version Version number of the log record format. The value 1 identifies this version.

flags Identifies presence of optional sections in the variable part of the record. The following bit settings are defined:

AP_TP_SUPERIOR_SECTION
When set, indicates a superior section is present.

SnD_type Node Data type. One of:

AP_TP_NONE
Log record does not represent a ready or committing transaction. *SldD_type* must have a value other than AP_TP_NONE.

AP_TP_READY_LOG
Log record represents a ready transaction node.

AP_TP_COMMIT_LOG
Log record represents a transaction node being committed.

SldD_type Log damage Data type. One of:

AP_TP_NONE
: No heuristic damage logged.

AP_TP_HEUR_MIX
: Heuristic mix exists in subtree.

AP_TP_HEUR_HAZ
: Heuristic hazard exists in subtree.

sub_count Number of subordinate branches present in record. The record contains a subordinate branch section for each of these branches. Zero if no subordinate branch sections are present.

The Node Details Section

This contains details of the node.

```
/*
 * node section -- always present
 */

<variable> ttnid      /* User assigned TTNID value      */
                      /* length == 0 if none assigned   */
<variable> aaid       /* Atomic Action Identifier       */
<variable> aet        /* Node's AE-title                */
<variable> rch        /* Node's Recovery Context Handle */
                      /* length == 0 if no handle       */
```

The fields *aaid*, *aet* and *rch* contain the ASN.1 BER encoding of their respective data types. They are defined as follows:

aaid

```
SEQUENCE {
    masters-name            CHOICE {
        name        [0] EXPLICIT AE-title,
        side        [1] ENUMERATED { sender (0), receiver (1) }
    },
    atomic-action-suffix    CHOICE {
        form1       [2] OCTET STRING,
        form2       [3] INTEGER
    }
}
```

aet

```
AE-title
```

rch

```
OCTET STRING
```

The Superior Branch Section

This contains details of the superior branch. This section is only present if the bit flag AP_TP_SUPERIOR_SECTION is set in the *flags* field of the header section.

```
/*
 * superior branch section -- present if superior set in flags
 */

<variable> branch_suffix    /* Superior Branch suffix */
<variable> aet              /* AE Title of superior */
<variable> rch              /* Recovery context handle provided */
                            /* by superior, length == 0 if none */
                            /* provided */
```

The fields *branch_suffix*, *aet* and *rch* contain the ASN.1 BER encoding of their respective data types. They are defined as follows:

branch_suffix

```
CHOICE {
    form1   [2] OCTET STRING,
    form2   [3] INTEGER
}
```

aet AE-title

rch OCTET STRING

The Subordinate Branch Section

There is one section for each subordinate branch. Each contains details of a single subordinate branch. If the *sub_count* field in the header section is zero, there are *no* subordinate branch sections present in the log record.

```
/*
 * subordinate branches section -- present if subordinate_count > 0
 * one section for each subordinate
 */
<integer> status            /* heuristic damage state. One of: */
                            /* - AP_TP_NONE */
                            /* - AP_TP_HEUR_MIX */
                            /* - AP_TP_HEUR_HAZ */
<variable> branch_suffix    /* Subordinate Branch Suffix */
<variable> aet              /* AE-title of subordinate */
<variable> rch              /* Recovery context handle provided by */
                            /* subordinate, length == 0 if none */
                            /* provided */
```

The fields *branch_suffix*, *aet* and *rch* contain the ASN.1 BER encoding of their respective data types. They are defined as follows:

branch_suffix

```
CHOICE {
    form1   [2] OCTET STRING,
    form2   [3] INTEGER
}
```

aet AE-title

rch OCTET STRING

Encoding and Decoding the Log Record

The following macros are used to encode and decode the log record fields:

AP_TP_VP_INT_LENGTH(val)
: Returns the number of octets that the encoding of the `<integer>` value *val* would occupy in the variable part.

AP_TP_VP_VAL_LENGTH(len)
: Returns the number of octets that the encoding of the data part of a `<variable>` value of length *len* would occupy in the variable part. Note that this excludes the amount of space necessary for the length item which precedes the actual data.

AP_TP_VP_ENCODE_INT(ptr, val)
: Encodes an `<integer>` value into the buffer location pointed at by *ptr* (which must be of type **unsigned char *****), and returns an updated pointer to the next available location in the buffer.

AP_TP_VP_ENCODE_VAL(ptr, len, val)
: Encodes a `<variable>` value into the buffer pointed at by *ptr* (which must be of type **unsigned char *****), and returns an updated pointer to the next available location in the buffer. Note that the preceding length item must have been encoded into the buffer using the macro AP_TP_VP_ENCODE_INT prior to this call.

AP_TP_VP_INT(ptr)
: Decodes the `<integer>` field or item from the buffer pointed at by *ptr* (which must be of type **unsigned char *****), and updates *ptr* to reference the next item in the buffer.

AP_TP_VP_PTR(ptr, len)
: Returns a pointer to the value part of a `<variable>` field in the buffer pointed at by *ptr* (which must be of type **unsigned char *****), and updates *ptr* to reference the next item in the buffer. Note that the user must have decoded the preceding length part of the field.

Encoding Implementation

The following encoding for the variable part balances minimal encoding with minimal processing overhead.

The representation close packs fields. There are no alignment or padding octets.

Integer fields are encoded into one, two or four octets depending on the integer value. Integer values ≤ 127 (0x7f) are encoded into a single octet, values ≤ 16383 (0x3fff) are encoded into two octets, and values above this are encoded into 4 octets.

The 0x80 bit of the first two octets is used to indicate continuation. If 0x80 bit is set in the first octet, the value is continue into the next octet. If the 0x80 bit is set in the second octet, the value is continued in the next two octets. The actual encoding of an integer becomes apparent from the example macros below.

Note this representation places an upper limit of (0x3fffffff) on an unsigned integer value.

Data fields are stored as is, with no padding or alignment.

The following are example macros to perform this encoding and decoding:

```
/*
 * return number of unsigned chars needed to hold an int field
 */
#define AP_TP_VP_INT_LENGTH(val) \
    (((val) ≤ 0x7f)? 1: ((val) ≤ 0x3fff)? 2: 4)

/*
 * return number of unsigned chars needed to hold
 * a variable data value
 */
#define AP_TP_VP_VAL_LENGTH(len) (len)

/*
 * encode an integer field and return pointer to next free location
 */
#define AP_TP_VP_ENCODE_INT(ptr, val) \
    (((val) ≤ 0x7f)? \
        *(ptr)++ = (unsigned char) (val):\
     ((val) ≤ 0x3fff)? \
        (*(ptr)++ = (unsigned char) (((val) & 0x7f) | 0x80),\
         *(ptr)++ = (unsigned char) ((val) >> 7) ) :\
     (*(ptr)++ = (unsigned char) (((val) & 0x7f) | 0x80),\
      *(ptr)++ = (unsigned char) ((((val) >> 7) & 0x7f) | 0x80),\
      *(ptr)++ = (unsigned char) (((val) >> 14) & 0xff),\
      *(ptr)++ = (unsigned char) (((val) >> 22) & 0xff)), (ptr))

/*
 * encode a variable data field and return pointer to next
 * free location
 */
#define AP_TP_VP_ENCODE_VAL(ptr, len, val) \
    (ptr = ((unsigned char *) memcpy((ptr), (val), (len))) + (len))

/*
 * return the integer field from the current location
 * referenced by ptr, and update ptr to reference the next field
 */
#define AP_TP_VP_INT(ptr) \
    (!((ptr)[0] & 0x80))? \
        (unsigned int) *(ptr)++: \
    (!((ptr)[1] & 0x80))? \
        ((ptr) += 2, ((unsigned int) (ptr)[-2] & 0x7f) | \
                     ((ptr)[-1] << 7)): \
    ((ptr) += 4, ((unsigned int) (ptr)[-4] & 0x7f) | \
                 (((ptr)[-3] & 0x7f) << 7) | \
                 ((ptr)[-2] << 14) | \
                 ((ptr)[-1] << 22))

/*
 * return ptr to variable length field and update ptr to
 * reference the next field
```

```
 */
#define AP_TP_VP_PTR(ptr, len) \
    ((ptr) += (len), ((ptr) - (len)))
```

2.14 Heuristic Logging

Heuristic damage logging requires some special discussion.

Recovery from heuristic conditions is outside the scope of ISO/IEC 10026, and for this reason it does not detail what information should be logged to enable diagnosis and rectification of such a condition.

When a heuristic hazard or heuristic mix condition is detected, XAP-TP updates the node's log record to include details of which branch experienced the condition, and passes the entire updated log record to the user in a TP_LOG_DAMAGE_IND primitive. In this way, XAP-TP ensures that the user has the opportunity to preserve enough basic information to diagnose which branches have an actual or potential heuristic condition.

The user should log this information. If the user chooses not to log all the information, it must at least preserve a minimum log damage record (see *TP_LOG_DAMAGE_IND* on page 182). If the user logs the entire record, once it is secure the previous log-ready, log-commit or log-damage record can be safely forgotten, as the new record combines the log-ready or log-commit information with the log-damage information. If, however, the user decides to log only a minimal damage record, it may forget a previous damage record for the node once the new record is secure.

On commit or rollback completion, the user may delete a log-ready or log-commit record immediately, but may only delete a record updated with log-damage status as a result of some local administration instruction. This ensures the log-damage record persists until administrative intervention diagnoses and rectifies any problem.

2.15 Instance State and Node State

XAP-TP maintains two state variables, one to reflect the state of an instance, and one to reflect the state of the current transaction node where multiple transaction nodes are being controlled from a single instance.

Instance State Attribute AP_STATE

The AP_STATE attribute value reflects the state of the library for an XAP-TP instance. It is used for two purposes:

- to reflect the state of progress through restart or resumption for an instance with the AP_TP_CONTROL category selected
- to reflect the state of any dialogue on the instance.

In the first case, once resume or restart has been completed, the instance will remain in the state AP_TP_IDLE unless the instance also has the AP_TP_DIALOGUE category selected, when it reflects the state of any dialogue on the instance.

Node State Attribute AP_TP_STATE

The AP_TP_STATE attribute is only present in the environment of an instance with the control category selected.

It reflects the state of the current transaction node identified by the AP_DTNID or AP_TTNID attribute. The state of a transaction node is taken to be that of one of the coordinated dialogues, or if there are none, one of the uncoordinated dialogues. If the transaction node is a heuristic damage entry from a completed transaction, AP_TP_STATE has the value AP_TP_HEURISTIC_LOG.

2.16 XAP-TP Instance Synchronisation

2.16.1 Principles for XAP-TP Instance Synchronisation

Synchronisation is only necessary for XAP-TP instances supporting coordinated dialogues.

Synchronisation is necessary under the following circumstances:

- when a failure occurs (an event which disrupts normal processing)
- when the commit instruction is received (TP-COMMIT indication)
- at commit (or rollback) completion.

The following conditions can initiate rollback:

- the XAP-TP user issuing a TP_ROLLBACK_REQ primitive on a control category instance
- the XAP-TP user issuing a TP_U_ABORT_REQ primitive for a dialogue
- loss of a single dialogue by TP_U_ABORT_IND or TP_P_ABORT_IND
- receipt of an explicit instruction to rollback from a remote transaction node (C-ROLLBACK-RI received), TP_ROLLBACK_IND
- a protocol error being detected
- a local error being detected by the XAP-TP provider.

Some further definitions are useful:

Active Coordinated Instance
: An open XAP-TP instance supporting a coordinated dialogue which has not been aborted.

Passive Coordinated Instance
: An Active coordinated instance which has had its dialogue aborted, but which is still participating in commitment or rollback of the current transaction (that is, awaiting a TP_COMMIT_COMPLETE_IND or TP_ROLLBACK_COMPLETE_IND).

The synchronisation which occurs depends on the state as described below.

States AP_TP_IDLE → AP_TP_WPREP_ALLreq_DATAP

Failure conditions causing rollback, which propagate TP_ROLLBACK_IND:

- issuing a TP_ROLLBACK_REQ
- receipt of a TP_ROLLBACK_IND.

Failure conditions causing rollback, which propagate TP_DIALOGUE_LOST_IND:

- receipt of a TP_U_ABORT_IND(rollback=TRUE)
- receipt of a TP_P_ABORT_IND(rollback=TRUE)
- issuing a TP_U_ABORT_REQ( )
- implicit or explicit close of the XAP-TP instance.

State AP_TP_PREPARING (20A)

Failure conditions causing rollback, which propagate TP_ROLLBACK_IND:

- issuing a TP_ROLLBACK_REQ
- receipt of a TP_ROLLBACK_IND.

Failure conditions causing rollback, which propagate TP_DIALOGUE_LOST_IND:

- receipt of TP_BEGIN_DIALOGUE_CNF(result=REJECTED, rollback=TRUE)
- receipt of a TP_U_ABORT_IND(rollback=TRUE)
- receipt of a TP_P_ABORT_IND(rollback=TRUE)
- implicit or explicit close of the XAP-TP instance.

State AP_TP_LOGGING_READY (20B)

Failure conditions not causing rollback, which propagate TP_DIALOGUE_LOST_IND:

- receipt of TP_P_ABORT_IND(rollback=FALSE)
- implicit or explicit close of the XAP-TP instance (at a Root node).

  Note: The user may have already commenced commitment.

Failure conditions causing rollback, which propagate TP_ROLLBACK_IND:

- issuing a TP_ROLLBACK_REQ
- receipt of a TP_ROLLBACK_IND.

Failure conditions causing rollback, which propagate TP_DIALOGUE_LOST_IND:

- receipt of a TP_U_ABORT_IND(rollback=TRUE)
- receipt of a TP_P_ABORT_IND(rollback=TRUE)
- implicit or explicit close of the XAP-TP instance (at an Intermediate or a Leaf node).

State AP_TP_WCOMMITind (20C)

Non-failure conditions, which propagate TP_COMMIT_IND:

- receipt of a TP_COMMIT_IND.

Failure conditions not causing rollback, which propagate TP_DIALOGUE_LOST_IND:

- receipt of TP_P_ABORT_IND(rollback=FALSE)
- implicit or explicit close of the XAP-TP instance.

Failure conditions causing rollback, which propagate TP_ROLLBACK_IND:

- receipt of a TP_ROLLBACK_IND.

Failure conditions causing rollback, which propagate TP_DIALOGUE_LOST_IND:

- receipt of a TP_U_ABORT_IND(rollback=TRUE).

State AP_TP_COMMIT_WDONEreq (21)

Failure conditions, which propagate TP_DIALOGUE_LOST_IND:

- receipt of TP_U_ABORT_IND(rollback=FALSE)
- receipt of TP_P_ABORT_IND(rollback=FALSE)
- implicit or explicit close of the XAP-TP instance.

State AP_TP_WCOMMIT_COMPind (22)

Failure conditions, which propagate TP_DIALOGUE_LOST_IND:

- receipt of TP_U_ABORT_IND(rollback=FALSE)
- receipt of TP_P_ABORT_IND(rollback=FALSE)
- implicit or explicit close of the XAP-TP instance.

Completion conditions, which propagate TP_COMMIT_COMPLETE_IND:

- receipt of TP_COMMIT_COMPLETE_IND.

State AP_TP_ROLL_WDONEreq (23)

Failure conditions, which propagate TP_DIALOGUE_LOST_IND:

- receipt of TP_BEGIN_DIALOGUE_CNF(result=REJECTED, rollback=FALSE)
- receipt of TP_U_ABORT_IND(rollback=FALSE)
- receipt of TP_P_ABORT_IND(rollback=FALSE)
- implicit or explicit close of the XAP-TP instance.

State AP_TP_WROLL_COMPind (24)

Failure conditions, which propagate TP_DIALOGUE_LOST_IND:

- receipt of TP_BEGIN_DIALOGUE_CNF(result=REJECTED, rollback=FALSE)
- receipt of TP_U_ABORT_IND(rollback=FALSE)
- receipt of TP_P_ABORT_IND(rollback=FALSE)
- implicit or explicit close of the XAP-TP instance.

Completion conditions, which propagate TP_ROLLBACK_COMPLETE_IND:

- receipt of TP_ROLLBACK_COMPLETE_IND.

2.16.2 Propagation

Transaction completion conditions (signified by TP_COMMIT_COMPLETE_IND or TP_ROLLBACK_COMPLETE_IND) are propagated to all active and passive coordinated instances and to the nominated control instance if this is not an active or passive coordinated instance.

For primitives other than TP_COMMIT_COMPLETE_IND and TP_ROLLBACK_COMPLETE_IND, the instance on which the primitive causing propagation is issued is known as the *originating instance*. Propagation takes place to all active coordinated instances of the node apart from the originating instance, and also to the nominated control instance of the node if it is neither an active coordinated instance nor the originating instance.

2.16.3 Implicit or Explicit Close of an XAP-TP Instance

Implicit or explicit close of an XAP-TP instance usually occurs as a result of some program or process error, which indicates the program or process can no longer drive the instance.

Recovery and resynchronisation for control instances is catered for by XAP-TP. Additional features will need to be added to XAP-TP to enable support for dialogue recovery when this is supported by OSI TP. Until that time, there is no mechanism to allow reconnection and resynchronisation of a (possibly reinstantiated) program to an individual dialogue.

Unfortunately the OSI TP specification does not model the loss of a part of the TPSUI, and so some special action is necessary in an XAP-TP provider to ensure correct functioning of OSI TP when an implicit or explicit close occurs.

Special care must be taken if the dialogue from the user has a TP_BEGIN_DIALOGUE_IND(confirmation=TRUE) indication outstanding when the XAP-TP instance supporting it is closed. The dialogue must be implicitly accepted before being aborted.

Implicit or explicit close of an XAP-TP instance with an uncoordinated dialogue extant causes the dialogue to be terminated as if a TP_U_ABORT_REQ had been issued.

When supporting a coordinated dialogue the action taken depends on the state of the node as follows:

States AP_TP_IDLE → AP_TP_WPREP_ALLreq_DATAP
: Treated as user issuing TP_U_ABORT_REQ.

  Note: This will cause rollback of the transaction node.

State AP_TP_PREPARING (20A)
: Causes dialogue to be aborted and transaction node to be rolled back.

State AP_TP_LOGGING_READY (20B)
: For an intermediate or leaf node, causes dialogue to be aborted and transaction node to be rolled back.

  For a root node treated as an allowable TP_U_ABORT_REQ, does not cause rollback (as commitment may already have been commenced by the user).

State AP_TP_WCOMMITind (20C)
: Treated as an allowable TP_U_ABORT_REQ, does not cause rollback.

State AP_TP_COMMIT_WDONEreq (21)
: Treated as an allowable TP_U_ABORT_REQ, does not cause rollback.

State AP_TP_WCOMMIT_COMPind (22)
: Treated as an allowable TP_U_ABORT_REQ, does not cause rollback, will cause transit to state AP_TP_COMMIT_WDONEreq.

State AP_TP_ROLL_WDONEreq (23)
: Treated as an allowable TP_U_ABORT_REQ, does not cause rollback.

State AP_TP_WROLL_COMPind (24)
: Treated as an allowable TP_U_ABORT_REQ, does not cause rollback, will cause transit to state AP_TP_ROLL_WDONEreq.

As an example, the state table intersect for TP_LOGGING_READY could be written as follows:

Nr, ^Danyb [VNfaT] [ABTPSUI] [NOTCHAIN] 20B
Nr, Danyb 20B
^Nr [DELIMIT] [ABTPSUI] [NOTCHAIN] [INITRB] [OWEDONE] 20B

2.16.4 TP_U_ABORT_REQ Primitives Issued by the User

A TP_U_ABORT_REQ primitive issued on an active coordinated instance during the active phase is a failure condition which causes the transaction to rollback. It is propagated to all active coordinated instances of the node apart from the originating instance, and also to the nominated control instance of the node if it is neither an active coordinated instance nor the originating instance. Note that when such a TP_U_ABORT_REQ is issued on a control and dialogue instance, no TP_DIALOGUE_LOST_IND is received and so the tp_fail_count for the subsequent TP_DONE_REQ must be set to the value one by the originating application. See Section 2.16.5, below.

A TP_U_ABORT_REQ primitive issued after a failure condition has been reported to the user is not itself treated as a failure condition. It is a legitimate action taken by the user in response to the reported failure (and therefore no propagation takes place for it).

A TP_U_ABORT_REQ primitive issued after a failure condition has been reported in state AP_TP_LOGIN_READY, behaves differently at a root node than at an intermediate or leaf node. At a root node the TP_U_ABORT_REQ does not cause rollback (because commitment may have been commenced by the user). At an intermediate or leaf node it causes the dialogue to be aborted and the transaction node to be rolled back.

2.16.5 Accounting for Failure Conditions

When a number of failure conditions have been signalled to the user and the user issues a TP_DONE_REQ primitive to acknowledge them, the OSI TP provider must be assured that all the outstanding failure conditions have been taken into account. To this end the provider maintains an internal variable for the transaction node to hold the count of outstanding error conditions (*tp_fail_count*).

tp_fail_count is set to zero at the start of each new transaction on the node, and is incremented each time one of the failure conditions described above occurs. The value of the node's *tp_fail_count* is returned on each of the following primitives when issued on an active coordinated instance:

- TP_P_ABORT_IND
- TP_U_ABORT_IND

- TP_BEGIN_DIALOGUE_CNF(REJECTED, . . .)

and unconditionally on the following primitives:

- TP_DIALOGUE_LOST_IND
- TP_ROLLBACK_IND
- TP_NODE_STATUS_IND.

The last value of *tp_fail_count* taken into account by the user must be passed on a TP_DONE_REQ primitive to enable the provider to check that no failure primitives are pending on the user. If no failure condition has been initiated or received by the user, the value zero should be used. If the user's *tp_fail_count* and the provider's *tp_fail_count* do not match, the TP_DONE_REQ primitive is refused with the error code [AP_TP_BADCD_FAIL_COUNT]. The TP_DONE_REQ primitive must be resubmitted when all pending failure indications have been taken into account. The failure count handling rules can be summarised as follows:

- If the fail count is zero when a TP_U_ABORT_REQ or TP_ROLLBACK_REQ is issued, then the fail count is incremented to 1.
- If the fail count is greater than zero when a TP_U_ABORT_REQ or TP_ROLLBACK_REQ is issued, then the fail count is not changed.
- A failure indication will always increment the fail count and the new fail count will be returned with the indication.
- TP_ROLLBACK_REQ will return TP_ROLLBACK_IND to all other active coordinated instances (not to the nominated control instance which originated the request).

2.16.6 Information Passed with TP_DIALOGUE_LOST_IND Primitive

The following information is passed with a TP_DIALOGUE_LOST_IND:

The argument *cdata→tp_options* indicates if the transaction, in which the recipient is involved, is being rolled back as a result of the failure, and whether the failure occurred on the dialogue from the superior. The bit values in *tp_options* used are:

AP_TP_ROLLBACK
: If set, the transaction is being rolled back. If unset, this dialogue loss has not commenced rollback.

AP_TP_SUPERIOR
: If set, the dialogue lost was that from the superior. If unset, the dialogue lost was to a subordinate.

The *cdata→tp_fail_count* argument holds the number of failure conditions which have occurred on the node since the start of the transaction, and must be used when issuing a TP_DONE_REQ primitive to acknowledge completion of any failure related actions.

2.16.7 Resuming Operation of a Control Instance

When resuming operation of a control category instance (see Section 2.12 on page 46), the user must determine the state of all transaction nodes assigned to the instance and take appropriate action.

A TP_NODE_STATUS_IND primitive is returned for each transaction node. In addition to returning the state of the node, this returns the current value of *tp_fail_count*. The user can determine from the node state whether a TP_DONE_REQ primitive is owed, and in conjunction with the software driving the coordinated dialogues determine appropriate action to take to restore itself to a consistent state.

2.16.8 Aligning the Commitment and Rollback State Tables

At the time of publication, the OSI TP service state tables do not treat the signalling of a dialogue failure to the user in a consistent manner for rollback and commitment.

In the rollback state 24 (AP_TP_WROLL_COMPind), a dialogue failure will change to state 23 (AP_TP_ROLL_WDONEreq) and a TP-DONE request will be expected. In the equivalent commitment state 22 (AP_TP_WCOMMIT_COMPind) no state change occurs. Although the state table will function correctly as defined, it is somewhat inconsistent, as the underlying OSI TP protocol state tables will not permit a TP-COMMIT-COMPLETE indication to be delivered to the user until the user has issued a TP-DONE request to acknowledge the outstanding failure, and the TPSP constraints on TP-COMMIT-COMPLETE indication in the service state this constraint.

If the OSI TP service state tables were employed with this limitation it would not be possible for the XAP-TP user to directly determine from the state whether a TP_DONE_REQ primitive was owed for the transaction node. This is particularly the case when the user resumes the use of a control instance.

In order to make the XAP-TP states directly reflect when a TP_DONE_REQ primitive is required, the following amendments to the OSI TP service state table for state 22 (AP_TP_WCOMMIT_COMPind) are used:

State	22	
Predicates	Dl	
Event		
TP-U-ABORT req		(blank cell)
TP-U-ABORT ind (Rollback="false")	^Db 21 [p][l]	
TP-P-ABORT ind (Rollback="false")	^Db 21 [p][l]	
TP-DONE req * (without Heuristic-Report)		(blank cell)

2.16.9 XAP-TP Instance Synchronisation on Global Primitives

When a global primitive is issued on a control instance, XAP-TP propagates the state change to AP_STATE on all other XAP instances supporting transaction mode dialogues of the transaction node. This allows each dialogue of the node to be in synchronisation with the actual state of the node. This prevents accidental rollback due to state inconsistencies between XAP-TP AP_STATE for a dialogue and the TP MACF state.

2.17 Advice on the Use of A-ABORT Request

It is not envisaged that the A-ABORT request service be used in conjunction with OSI TP except under unusual circumstances. For example, the following are two legitimate uses of A-ABORT request with OSI TP:

- The user has allocated an association for dialogue establishment, is part way through sending a fragmented TP_BEGIN_DIALOGUE_REQ primitive, and due to a failure can no longer complete the user data portion of the primitive. Rather than allow an invalidly constructed PDU to be sent, the user can issue an A_ABORT_REQ primitive to abandon the association and so discard the erroneous PDU.
- The user has received a U-ASE pdu which is invalidly encoded, and cannot therefore be sensibly decoded. The user can signal this U-ASE protocol error by issuing an A_ABORT_REQ to abandon the association. Alternatively, the user may consider discarding the pdu and issuing a TP_U_ERROR_REQ primitive.

If the user does make use of the A-ABORT request service to abandon the association underlying a dialogue, they should be aware that OSI TP interprets any user data passed as TP pdus, and the presence of anything user generated in the user data field will result in OSI TP signalling a protocol error on receipt and decoding of the A-ABORT pdu. For this reason the user shall not pass user data on an A-ABORT request primitive.

2.18 Advice on Flushing the Concatenator

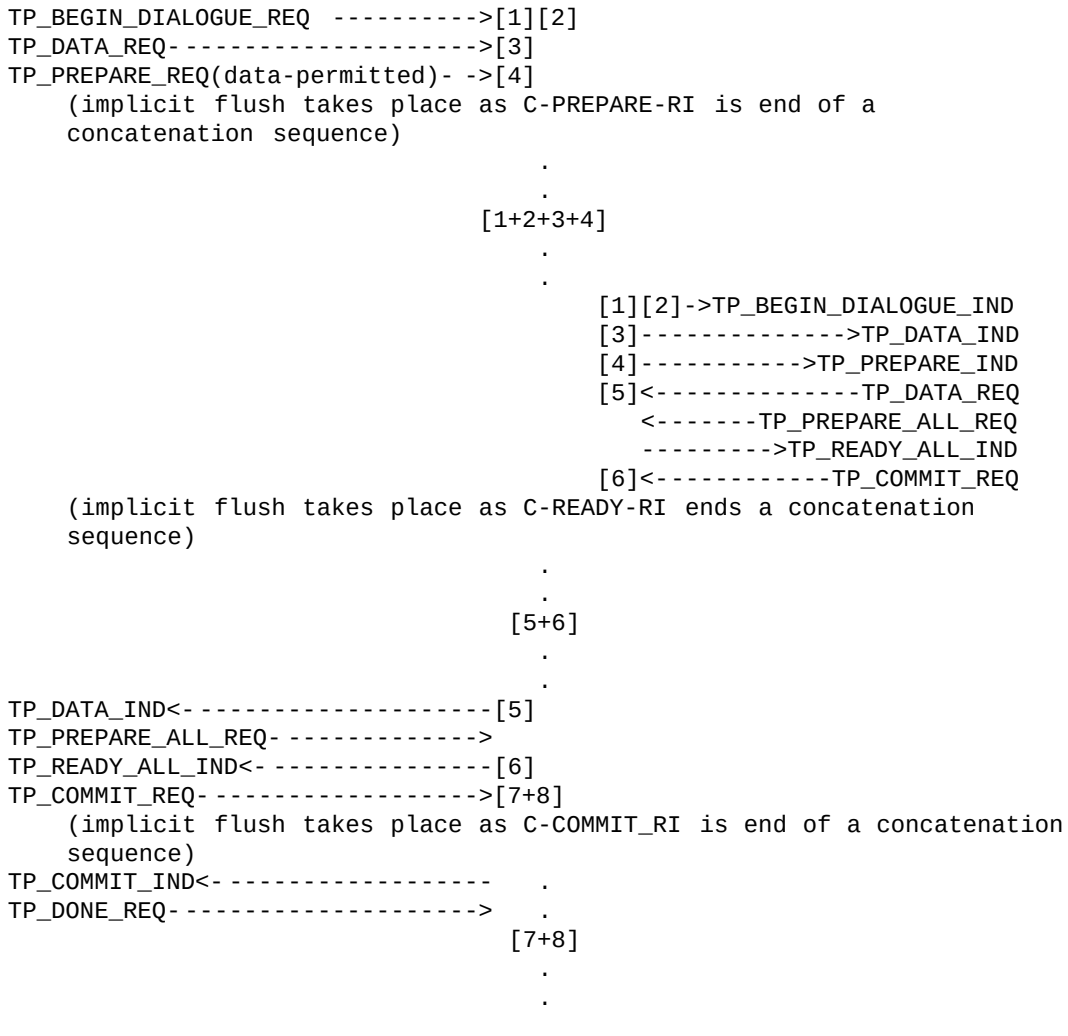
Concatenation in OSI TP provides a powerful tool to optimise network traffic for application exchanges. The OSI TP and CCR concatenation rules define the permissible concatenation sequences. The user can rely on the concatenation rules of TP and CCR to flush the concatenator at the end of a concatenation sequence. Alternatively, the user can flush the contents of the concatenator prior to the end of a concatenation sequence by setting the AP_FLUSH bit in the *flags* field of an *ap_snd*( ) for a primitive, or by issuing an explicit *ap_snd*( ) of a TP_FLUSH_REQ primitive.

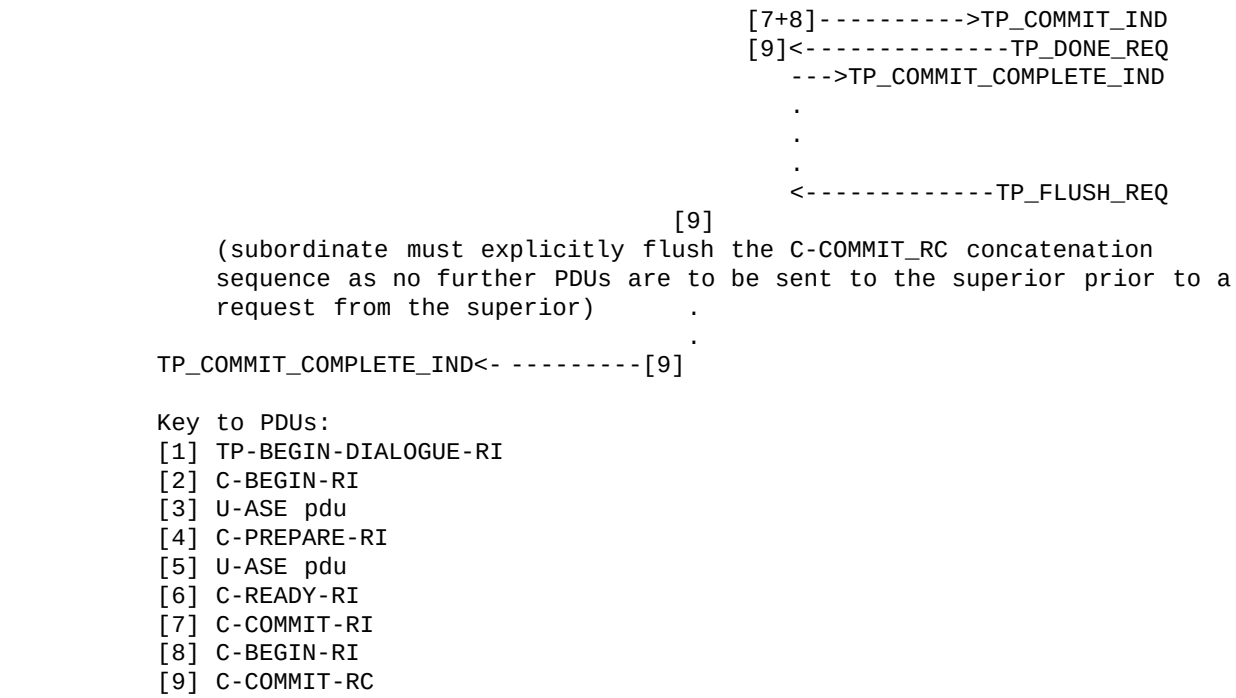
There are situations where the OSI TP implementation will implicitly flush the contents of the concatenator prior to end of a concatenation sequence. These are:

- when the dialogue ceases to exist
- in polarised control, for a subordinate at commit completion when control resides with the superior after commitment.

The user should therefore be aware of concatenation sequences under construction and flush them when the PDUs are required to be delivered to the peer.

For example, the following exchanges show a shared control chained transactions dialogue, where the user must explicitly flush the concatenator in order to complete commitment.





Note that the subordinate must issue an explicit flush in order to clear the concatenator of the C-COMMIT-RC concatenation sequence

Of course, if the subordinate already had U-ASE PDUs to send, it could issue these requests prior to flushing the concatenator, and thereby optimise network traffic further.

2.19 Using the XAP-TP Interface

Below is a summary of the steps required to establish a dialogue with a remote application entity using XAP-TP. The procedure presented is intended solely as a general description of how the interface might be used. It should not be construed as an attempt to provide a template for constructing any particular application. Moreover, it is assumed that the service user is familiar with the OSI TP protocol and understands the role of the OSI TP-service-user in establishing, using and terminating a dialogue between two application entities.

Obtain an XAP Instance

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

Initialise the XAP-TP Environment

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

After the environment is initialised, the user may examine or alter the environment attributes, subject to the readability and writability restrictions discussed on the *ap_env*( ) manual page.

Select TP MODE

Before using any of the XAP-TP extensions, the XAP instance must be placed into AP_TP_MODE. This is accomplished by setting the AP_MODE_SEL attribute to AP_TP_MODE by a call to the *ap_set_env*( ) function or a call to the *ap_init_env*( ) function.

Select the Category of TP Primitives

Before any service primitives can be sent or received, the categories of TP primitives to be available on the XAP instance must be selected. In this case, the AP_TP_DIALOGUE category of primitives is needed. This is accomplished by setting the AP_TP_CATEGORY environment attribute to AP_TP_DIALOGUE by a call to the *ap_set_env*( ) function or a call to the *ap_init_env*( ) function.

Bind the XAP-TP Instance

Before any service primitives can be sent or received, the XAP instance must be bound to a local AET, a local TPSUT and a recovery context group selected (if required). This is accomplished by setting the AP_BIND_TPADDR and AP_LCL_TPSUT attributes to values which the service user is authorised to use, and the AP_URCH attribute to the user-assigned recovery context handle (if required), and then calling *ap_bind*( ) to validate the address. These attributes can be set using either a call to the *ap_set_env*( ) function or a call to the *ap_init_env*( ) function.

Set Other Environment Attributes

In addition to AP_BIND_TPADDR and AP_LCL_TPSUT, other environment variables must be set before a dialogue is established, particularly if the service user is the dialogue-initiator. For example, before issuing a TP_BEGIN_DIALOGUE_REQ primitive, the service user must specify the address of the remote entity by setting the AP_REM_APT, AP_REM_APID, AP_REM_AEQ, AP_REM_AEID and AP_REM_TPSUT attributes, and set the AP_CNTX_NAME attribute to select an application context for the dialogue.

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

Send or Receive TP_BEGIN_DIALOGUE Service Primitives

Once the XAP-TP environment has been properly initialised, the service user may attempt to establish a dialogue with a remote TPSUI by using the TP_BEGIN_DIALOGUE service primitives in conjunction with the *ap_snd*( ) and *ap_rcv*( ) functions. Each of the TP_BEGIN_DIALOGUE primitives is described in detail in Chapter 7.

Transfer Data

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

Releasing the Dialogue

A dialogue may be released either normally (using the TP_END_DIALOGUE or TP_DEFERRED_END_DIALOGUE primitives), or abnormally (using the TP_U_ABORT primitive).

2.20 Summary

This chapter provided an introduction to the XAP-TP interface. Chapter 4 contains information about all of the XAP-TP environment attributes and functions, where these are different from those in the **XAP** specification. Chapter 7 provides information that is specific to each of the XAP-TP service primitives.

Chapter 3

Environment

This chapter specifies the additional and changed XAP environment attributes for OSI TP.

The following subset (only) of the XAP environment attributes are available unchanged within the XAP-TP environment. These attributes are:

AP_CNTX_NAME
AP_DCS
AP_FLAGS
AP_MSTATE
AP_QOS
AP_LIB_AVAIL
AP_LIB_SEL

The XAP-TP environment is made up of the XAP environment plus an additional set of TP related attributes. These additional attributes are used by XAP-TP to keep TP-specific state information and to hold the various pieces of data required to establish and maintain a dialogue with another TPSUI and to commit or rollback transactions.

Three existing attributes, AP_MODE_AVAIL, AP_MODE_SEL and AP_STATE, have modified semantics in XAP-TP. An XAP-TP implementation may provide access to all XAP-TP modes through a single provider, or alternatively provide access to subsets of the available modes through multiple service providers.

The attributes with altered semantics are:

AP_MODE_AVAIL
The AP_MODE_AVAIL attribute indicates the available modes of operation for XAP and provider. The modes that may be supported are those of XAP plus TP mode (AP_TP_MODE). See also the XAP definition of AP_MODE_AVAIL.

AP_MODE_SEL
The AP_MODE_SEL attribute indicates the mode in which XAP and provider are to be used. Only one mode can be selected at a time. A provider which supports only TP mode will have this attribute set to AP_TP_MODE as default.

AP_ROLE_ALLOWED
The AP_ROLE_ALLOWED attribute is used to specify how an XAP-TP instance may be used. Specifically, the value of this attribute indicates whether the XAP-TP instance may be used to send a TP_BEGIN_DIALOGUE_REQ primitive, receive a TP_BEGIN_DIALOGUE_IND primitive, or both.

Notes:

1. This attribute only affects the roles that an XAP-TP instance may play with respect to dialogue establishment. Changing the value of this attribute will not affect the way in which XAP-TP participates in a dialogue that has already been established.

2. A a TP_BEGIN_DIALOGUE_IND primitive may be received event after this attribute has been set to AP_INITIATOR in the AP_TP_IDLE state. This situation occurs when AP_ROLE_ALLOWED included the value AP_RESPONDER and a TP_BEGIN_DIALOGUE_IND had been queued prior to setting AP_ROLE_ALLOWED to AP_INITIATOR. To avoid this

situation, the AP_ROLE_ALLOWED attribute should be set before the XAP-TP instance is bound to an address if it will only be used to initiate dialogues.

AP_ROLE_CURRENT
The AP_ROLE_CURRENT attribute indicates the role of the user in the current dialogue. The attribute is set to AP_INITIATOR as soon as a TP_BEGIN_DIALOGUE_REQ is sent, and remains unchanged until the dialogue is rejected or terminated. Similarly, the attribute is set to AP_RESPONDER upon receipt of a TP_BEGIN_DIALOGUE_IND and remains unchanged until the dialogue is terminated.

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

In XAP-TP, there are a set of attributes which apply to the node in the dialogue or transaction tree rather than to a specific dialogue. These are referred to as global attributes, and are accessible through the environment of each instance tied to the node.

Setting the value of a global attribute through the environment of one instance changes the value for all instances tied to the node.

The additional global TP attributes in the XAP-TP environment are:

AP_AAID
The AP_AAID attribute holds the Atomic Action Identifier of the transaction on the TPSUIs node in the transaction tree. If none of the dialogues of the TPSUI are in transaction mode, the attribute holds the Atomic Action Identifier to be used for the next transaction to be commenced by the TPSUI.

AP_NEXT_AAID
The AP_NEXT_AAID attribute holds the Atomic Action Identifier to be used for the next transaction on the TPSUIs node in the transaction tree when chained transactions are in use. If unset, a new AAID will be generated by the TP service provider.

When the next transaction in a chain starts at a root node, the value in this attribute is used as the Atomic Action Identifier for the transaction. It is copied to the AP_AAID attribute and this attribute is unset. If the attribute is not set, the OSI TP implementation will generate a new Atomic Activity Identifier and place it in the AP_AAID attribute.

AP_NEXT_TTNID
The AP_NEXT_TTNID holds the next Transaction Tree Node Identifier to be used to identify the next transaction in a chain when chained transactions are in use.

When the next transaction in a chain commences at a node, the value of this attribute is copied to the AP_TTNID attribute and this attribute is unset. If chained transactions are in use, this attribute must be set prior to issuing a TP_COMMIT_REQ primitive for commitment and prior to issuing a TP_DONE_REQ primitive during rollback.

AP_TP_STATE
The AP_TP_STATE attribute is used to convey state information about the single transaction node identified by the AP_TTNID or AP_DTNID attribute. It is used to determine which control category primitives are legal.

Note: If the AP_TTNID or AP_DTNID does not identify a transaction node, its value is undefined. This attribute only exists in the environment of XAP-TP instances with the AP_TP_CONTROL category selected.

AP_TTNID
The AP_TTNID attribute holds the Transaction Tree Node Identifier of the TPSUI. Setting

the value of the AP_TTNID attribute on an endpoint which does not have AP_TP_DIALOGUE set in AP_TP_CATEGORY, will have the effect of setting the value of the AP_DTNID attribute to *not present*.

The additional local TP attributes in the XAP-TP environment are:

AP_BIND_TPADDR
The AP_BIND_TPADDR is used to indicate the address to which this instance of XAP is *bound*.

Setting this attribute and calling *ap_bind*( ) will result in the attributes *AP_LCL_APT*, *AP_LCL_AEQ*, *AP_LCL_APID* and *AP_LCL_AEID* being set, and the set of allowable local TPSUTs being declared for a RESPONDER. A *listener* application may use a *wildcard* for the set of local TPSUTs and use the called TPSUT (conveyed in *AP_LCL_TPSUT*) to dispatch dialogues.

AP_BRID
The BRID attribute holds the Branch Identifier of the transaction branch on the instance. If the instance is not supporting a transaction branch, it holds the branch identifier to be used for the next transaction branch on the instance.

AP_CONTROL_ID
The AP_CONTROL_ID attribute uniquely identifies a control instance within a recovery context group within an AET.

AP_DTNID
The AP_DTNID attribute indicates to which node of the dialogue tree a dialogue on this instance belongs. Setting the value of the AP_DTNID attribute on an endpoint which does not have AP_TP_DIALOGUE set in AP_TP_CATEGORY will have the effect of setting the value of the AP_TTNID attribute to *not present*.

AP_LCL_AEID
The AP_LCL_AEID attribute holds the Application Entity Invocation Identifier of the TPSUI.

This attribute is not directly settable by the user. However, setting the AP_BIND_TPADDR attribute will result in this attribute being set to the AEID portion of that attribute.

AP_LCL_AEQ
The AP_LCL_AEQ attribute holds the Application Entity Qualifier of the TPSUI.

This attribute is not directly settable by the user. However, setting the AP_BIND_TPADDR attribute will result in this attribute being set to the AEQ portion of that attribute.

AP_LCL_APID
The AP_LCL_APID attribute holds the Application Process Invocation Identifier of the TPSUI.

This attribute is not directly settable by the user. However, setting the AP_BIND_TPADDR attribute will result in this attribute being set to the APID portion of that attribute.

AP_LCL_APT
The AP_LCL_APT attribute holds the Application Process Title of the TPSUI.

This attribute is not directly settable by the user. However, setting the AP_BIND_TPADDR attribute will result in this attribute being set to the APT portion of that attribute.

AP_LCL_TPSUT
The AP_LCL_TPSUT attribute holds the TP service User Title of the TPSUI.

AP_NEXT_BRID
The AP_NEXT_BRID attribute holds the Branch Identifier to be used for the next transaction branch on the dialogue when chained transactions are in use. If unset, the existing Branch Identifier is reused for the next transaction branch.

When the next transaction branch in a chain starts on a dialogue, the value in this attribute is used as the Branch Identifier. It is copied to the AP_BRID attribute and this attribute is unset. If the attribute is not set, the existing Branch Identifier in AP_BRID will be reused for the new transaction branch.

AP_REM_AEID
The AP_REM_AEID attribute holds the Application Entity Invocation Identifier of the remote TPSUI.

AP_REM_AEQ
The AP_REM_AEQ attribute holds the Application Entity Qualifier of the remote TPSUI.

AP_REM_APID
The AP_REM_APID attribute holds the Application Process Invocation Identifier of the remote TPSUI.

AP_REM_APT
The AP_REM_APT attribute holds the Application Process Title of the remote TPSUI.

AP_REM_TPSUT
The AP_REM_TPSUT attribute holds the TP service User Title of the remote TPSUI.

AP_TPFU_AVAIL
The AP_TPFU_AVAIL attribute indicates which TP functional units are currently available.

AP_TPFU_SEL
The AP_TPFU_SEL attribute indicates which TP functional units have been requested for use on the current instance.

AP_TP_AVAIL
The AP_TP_AVAIL attribute indicates which versions of the TP protocol are currently available.

AP_TP_CATEGORY
The AP_TP_CATEGORY attribute holds the categories of TP primitives which are usable on the instance.

AP_TP_COPYENV
The AP_TP_COPYENV attribute is used to indicate whether certain environment attributes that correspond to parameters on the TP-BEGIN-DIALOGUE and TP-BEGIN-TRANSACTION indications, or to parameters on commit or log category indications, should be returned to the user in the *cdata* argument of the *ap_rcv*( ) function. When the value of the attribute is TRUE, these attributes are returned via the *cdata* argument.

AP_TP_SEL
The AP_TP_SEL attribute indicates which version of the TP protocol has been selected for use on the current instance.

AP_URCH
The AP_URCH uniquely identifies a group of transaction nodes within an AET for recovery purposes.

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

Attribute Name	Affects	Is Affected By
AP_BIND_TPADDR	AP_LCL_APT AP_LCL_AEQ AP_LCL_APID AP_LCL_AEID	
AP_LCL_APT		AP_BIND_TPADDR
AP_LCL_AEQ		AP_BIND_TPADDR
AP_LCL_APID		AP_BIND_TPADDR
AP_LCL_AEID		AP_BIND_TPADDR
AP_DTNID	AP_TTNID	AP_TTNID
AP_TTNID	AP_DTNID	AP_DTNID

The table on the following pages provides additional information about the XAP-TP environment attributes. The following data is included:

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

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

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

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

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

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

Attribute	Type/Values	Readable	Writable
AP_AAID	ap_aaid_t default: not present	always	any state whilst node not in a transaction
AP_BIND_TPADDR	ap_tpaddr_t default: none	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_BRID	ap_brid_t default: not present	always	any state whilst dialogue not in a transaction
AP_CONTROL_ID	ap_cid_t default: not present	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_DTNID	ap_dtnid_t default: not present	always	only in state AP_TP_IDLE
AP_LCL_AEID	ap_aei_api_id_t	always	never
AP_LCL_AEQ	ap_aeq_t	always	never
AP_LCL_APID	ap_aei_api_id_t	always	never
AP_LCL_APT	ap_apt_t	always	never

AP_LCL_TPSUT	ap_tpsut_t default: not present	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_MODE_AVAIL	unsigned long bit values: AP_NORMAL_MODE AP_X410_MODE AP_TP_MODE	always	never
AP_MODE_SEL	unsigned long bit values: AP_NORMAL_MODE AP_X410_MODE AP_TP_MODE default: AP_NORMAL_MODE if available AP_TP_MODE otherwise	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_NEXT_AAID	ap_aaid_t default: not present	always	All except: AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind, AP_TP_WROLL_COMPind AP_TP_ZOMBIE AP_TP_WRESUMEreq AP_TP_RESUME, AP_TP_WRESTARTreq AP_TP_RESTART AP_TP_W_RESTART_COMPLETEind
AP_NEXT_BRID	ap_brid_t default: not present	always	All except: AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind, AP_TP_WROLL_COMPind AP_TP_ZOMBIE AP_TP_WRESUMEreq AP_TP_RESUME, AP_TP_WRESTARTreq AP_TP_RESTART AP_TP_W_RESTART_COMPLETEind
AP_NEXT_TTNID	ap_ttnid_t default: not present	always	All except: AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind, AP_TP_WROLL_COMPind AP_TP_ZOMBIE AP_TP_WRESUMEreq AP_TP_RESUME, AP_TP_WRESTARTreq AP_TP_RESTART AP_TP_W_RESTART_COMPLETEind

Attribute	Type/Values	Readable	Writable
AP_REM_AEQ	ap_aeq_t default: not present	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_REM_APID	ap_aei_api_id_t default: not present	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_REM_APT	ap_apt_t default: not present	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_REM_TPSUT	ap_tpsut_t default: not present	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_STATE	unsigned long one of: AP_TP_UNBOUND AP_TP_IDLE AP_TP_WALLOCcnf AP_TP_ALLOCATED AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDcnf AP_TP_WENDrsp AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_PREPARING AP_TP_LOGGING_READY AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind AP_TP_ZOMBIE AP_TP_WRESUMEreq AP_TP_RESUME AP_TP_WRESTARTreq AP_TP_RESTART AP_TP_WRESTART_COMPLETEind		

Attribute	Type/Values	Readable	Writable
AP_TPFU_AVAIL	unsigned long bit values: AP_TP_POLARIZED_CONTROL AP_TP_SHARED_CONTROL AP_TP_COMMIT_AND_CHAINED AP_TP_COMMIT_AND_UNCHAINED AP_TP_HANDSHAKE	always	never
AP_TPFU_SEL	unsigned long bit values: AP_TP_POLARIZED_CONTROL AP_TP_SHARED_CONTROL AP_TP_COMMIT_AND_CHAINED AP_TP_COMMIT_AND_UNCHAINED AP_TP_HANDSHAKE default: NULL	always	only in states AP_TP_UNBOUND AP_TP_IDLE
AP_TP_AVAIL	unsigned long bit values: AP_TPVER1	always	never
AP_TP_CATEGORY	unsigned long bit values: AP_TP_DIALOGUE AP_TP_CONTROL	always	AP_TP_UNBOUND
AP_TP_COPYENV	long one of: TRUE FALSE default: FALSE	always	always
AP_TP_SEL	unsigned long bit values: AP_TPVER1 default: AP_TPVER1	always	only in states AP_TP_UNBOUND AP_TP_IDLE

Attribute	Type/Values	Readable	Writable
AP_TP_STATE	unsigned long one of: AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDcnf AP_TP_WENDrsp AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_PREPARING AP_TP_LOGGING_READY AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind AP_TP_ZOMBIE AP_TP_HEURISTIC_LOG	always	never
AP_TTNID	ap_ttnid_t default: not present	always	always
AP_URCH	ap_urch_t default: not present	always	only in states AP_TP_UNBOUND AP_TP_IDLE

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

ap_aaid_t is used to convey objects specified as ASN.1 type ATOMIC-ACTION-IDENTIFIER and is defined as:

```
typedef struct {
    int size;              /* buffer size in bytes */
    unsigned char *udata;  /* buffer with user-encoded */
                           /* ATOMIC-ACTION-IDENTIFIER */
} ap_aaid_t;
```

udata is a pointer to a buffer of user encoded ATOMIC-ACTION-IDENTIFIER; *size* is the length of that buffer in octets.

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

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

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

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

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

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

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

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

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

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

ap_brid_t is used to convey objects specified as ASN.1 type BRANCH-IDENTIFIER and is defined as:

```
typedef struct {
    int size;              /* buffer size in bytes */
    unsigned char *udata;  /* buffer with user encoded BRANCH-IDENTIFIER */
} ap_brid_t;
```

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

ap_cid_t is used to convey control identifier values, and is defined as:

```
typedef struct {
    int size;             /* control identifier length in bytes */
    unsigned char *udata; /* buffer with control identifier */
} ap_cid_t;
```

udata is a pointer to a buffer containing the control identifier; *size* is the length of that buffer in octets.

ap_dtnid_t is used to convey Dialogue Tree Node Identifiers and is defined as:

```
typedef struct {
    int size;              /* buffer size in bytes */
    unsigned char *udata;  /* buffer with Dialogue Tree Node Identifier */
} ap_dtnid_t;
```

udata is a pointer to a buffer containing a Dialogue Tree Node Identifier; *size* is the length of that buffer in octets.

ap_urch_t is used to convey the user-assigned recovery context handle to which an instance is bound. It is defined as:

```
typedef struct {
    int size;              /* recovery context handle length in bytes */
    unsigned char *udata;  /* buffer with user-assigned recovery */
                           /* context handle */
} ap_urch_t;
```

udata is a pointer to a buffer containing the user-assigned recovery context handle; size is the length of that buffer in octets, in the range 0 to 32.

Note: The value in the buffer is *not* encoded (has no tag and length wrapping).

ap_tpaddr_t is used to convey the address to which an instance is bound and to declare the allowable set of local TPSUTs for a responder. It is defined as:

```
typedef struct {
    ap_apt_t apt;          /* application process title */
    ap_aei_api_id_t apid;  /* application process invocation identifier */
    ap_aeq_t aeq;          /* application entity qualifier */
    ap_aei_api_t aeid;     /* application entity invocation identifier */
    long n_tpsuts;         /* number of TP service user titles */
    ap_tpsut_t *tpsuts;    /* array of TP service user titles */
} ap_tpaddr_t;
```

apt and *aeq* give the application entity title. *apid* and *aeid* give the application process invocation identifier and application entity invocation identifiers, respectively.

The *n_tpsuts* element is used to specify the number of tpsut components in the array *tpsuts*. Each element in the array *tpsuts* holds a TP service user title.

When used to represent a wildcard TP service user title, the value of *n_tpsuts* shall be zero. In this case all locally-defined TP service user titles are implied. In all other cases, *n_tpsuts* must be positive.

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

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

udata is a pointer to a buffer of user-encoded TPSU-title; *size* is the length of that buffer in octets. TPSU-title values are stored in their encoded form, including *tag* and *length*.

For optional PDU parameters of type TPSU-title, setting *size* to –1 indicates that the parameter is not present. In an *ap_tpaddr_t* structure, an element in the *tpsuts* array having a *size* of –1 indicates an omitted TPSU-title. In addition, an optional parameter that corresponds to an environment attribute may be specified to be absent by invoking *ap_set_env*( ) with a NULL pointer as the *val* argument.

ap_ttnid_t is used to convey locally-defined Transaction Tree Node Identifiers and is defined as:

```
typedef struct {
    int size;              /* buffer size in bytes */
    unsigned char *udata;  /* buffer transaction tree node identifier */
} ap_ttnid_t;
```

udata is a pointer to a buffer containing a user-defined local transaction tree node identifier; *size* is the length of that buffer in octets.

Values are assigned to attributes either through *ap_init_env*( ), *ap_set_env*( ) or during *ap_rcv*( ) event processing. When *ap_get_env*( ) is issued for an attribute that has no value assigned, the location pointed at by *aperrno_p* is set to [AP_BADENV].

Chapter 4

XAP-TP Functions

This chapter presents the manual page definitions for the XAP-TP SPI. These define the functions which make up XAP-TP, providing the detailed specifications of parameters and data structures where there are differences from the corresponding XAP functions.

Manual pages which have not changed from the **XAP** specification are not repeated here.

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

4.1 Overview

4.1.1 Functions

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

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

4.1.2 Errors

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

Each function description includes a list of error conditions that are reported by the XAP-TP Library. In addition, errors reported by underlying service providers (that is, OSI TP, ACSE, Presentation, Session, Transport, ASN.1 or the operating system) may be passed through to the user. The *class* of a particular error can be determined by examining the two most significant octets of the error code. The following bit masks can be used to distinguish between the various error classes:

```
/*
 * These ID numbers for each protocol are used to distinguish
 * #defines of various kinds for each layer, such as primitive
 * names, environment attribute names, error codes, and so on.
 *
 * ID NUMBERS MAY NOT EXCEED 125 (could be sign extension problems
 * otherwise).
 */
#define AP_TP_ID   (15)
#define AP_ASN1_ID (11)
#define AP_ID      (8)
#define AP_ACSE_ID (7)
#define AP_PRES_ID (6)
#define AP_SESS_ID (5)
#define AP_TRAN_ID (4)
#define AP_OS_ID   (0)
```

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

XAP-TP Library Errors

[AP_ACCES]
Request to bind to specified address denied.

[AP_AGAIN]
Request not completed.

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

[AP_BADATTRVAL]
Bad value for environment attribute.
cdata field value invalid: *act_id*.

[AP_BADCD_DIAG]
cdata field value invalid: *diag*.

[AP_BADCD_EVT]
cdata field value invalid: *event*.

[AP_BADCD_OLD_ACT_ID]
cdata field value invalid: *old_act_id*.

[AP_BADCD_OLD_CONN_ID]
cdata field value invalid: *old_conn_id*.

[AP_BADCD_RES]
cdata field value invalid: *res*.

[AP_BADCD_RESYNC_TYPE]
cdata field value invalid: *resync_type*.

[AP_BADCD_RSN]
cdata field value invalid: *rsn*.

[AP_BADCD_SYNC_P_SN]
cdata field value invalid: *sync_p_sn*.

[AP_BADCD_SYNC_TYPE]
cdata field value invalid: *sync_type*.

[AP_BADCD_TOKENS]
cdata field value invalid: *tokens*.

[AP_BADDATA]
User data not allowed on this service.

[AP_BADENC]
Bad encoding choice in enveloping function.

[AP_BADENV]
A mandatory attribute is not set.

[AP_BADF]
Not a presentation service endpoint.

[AP_BADFLAGS]
The specified combination of flags is invalid.

[AP_BADFREE]
Could not free structure members.

[AP_BADKIND]
Unknown structure type.

[AP_BADPARSE]
Attribute parse failed.

[AP_BADPRIM]
Unrecognised primitive from user.

[AP_BADREF]
Bad reference in enveloping function.

[AP_BADRESTR]
Attributes not restored due to more bit on.

[AP_BADROLE]
Request invalid due to value of AP_ROLE.

[AP_BADSAVE]
Attributes not saved due to more bit on.

[AP_BADSAVEF]
Invalid FILE pointer.

[AP_BADLSTATE]
Instance in bad state for that command.

[AP_BADUBUF]
Bad length for user data.

[AP_DATA_OVERFLOW]
User data and presentation pci exceeds 512 bytes.

[AP_HANGUP]
Association closed or aborted.

[AP_LOOK]
A pending event requires attention.

[AP_NOATTR]
No such attribute.

[AP_NOBUF]
Could not allocate enough buffers.

[AP_NODATA]
An attempt was made to send a primitive which requires user data without user-data.

[AP_NOENV]
No environment for that fd.

[AP_NOMEM]
Could not allocate enough memory.

[AP_NOREAD]
Attribute is not readable.

[AP_NOT_SUPPORTED]
The action requested is not supported by this implementation of XAP-TP.

[AP_NO_PROVIDER]
The XAP-TP service provider servicing this instance has become unavailable. The XAP-TP instance has reverted to the AP_TP_UNBOUND state.

[AP_NOWRITE]
Attribute is not writable.

[AP_PDUREJ]
Invalid PDU rejected.

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

[AP_TP_BADCD_FAIL_COUNT]
The given *tp_fail_count* does not match the XAP-TP provider's count of failure conditions.

[AP_TP_BADCD_TP_OPTIONS]
cdata field value invalid: *tp_options*.

[AP_TP_BAD_CONTROL_ID]
The given control identifier is already in use on another XAP-TP instance within the recovery context group.

[AP_TP_BAD_LOG]
The log record supplied does not belong to AE-title the instance is bound to, or does not belong to the recovery context group the instance is bound to.

[AP_TP_BAD_NODE]
AP_TTNID or AP_DTNID does not identify an extant transaction node.

[AP_TP_BAD_URCH]
The node does not belong to the same recovery context group as the XAP-TP instance.

[AP_TP_BAD_TTNID]
The TTNID in the log record supplied is already in use by another transaction node in the recovery context group.

[AP_TP_BAD_UDATA]
User data not allowed when there is no association already established.

[AP_TP_RESTART_REQD]
The recovery context group is currently unavailable.

[AP_TP_RESTARTING]
The recovery context group is currently restarting.

4.1.3 Mapping between XAP-TP and OSI TP Service State Numbers

The XAP-TP states are strongly linked to the OSI TP service state numbers. This relationship is shown in Table 4-1.

XAP-TP State	OSI TP Service State
AP_TP_IDLE	1
AP_TP_ALLOCATED	-
AP_TP_WALLOCcnf	-
AP_TP_DATA_XFER	2
AP_TP_RECV	3
AP_TP_ERROR_RECV	4
AP_TP_ERROR	5
AP_TP_WHANDcnf	6
AP_TP_WHANDrsp	7
AP_TP_WHANDrsp_WHANDcnf	8
AP_TP_WHANDcnf_WENDrsp	9
AP_TP_WHANDrsp_WENDcnf	10
AP_TP_WENDcnf	11
AP_TP_WENDrsp	12
AP_TP_WHAND_GCcnf	13
AP_TP_WHAND_GCrsp	14
AP_TP_WREADYind	15
AP_TP_WREADYind_DATAP	16
AP_TP_READY	17
AP_TP_WPREP_ALLreq	18
AP_TP_WPREP_ALLreq_DATAP	19
AP_TP_PREPARING	-
AP_TP_LOGGING_READY	-
AP_TP_WCOMMITind	20
AP_TP_COMMIT_WDONEreq	21
AP_TP_WCOMMIT_COMPind	22
AP_TP_ROLL_WDONEreq	23
AP_TP_WROLL_COMPind	24
AP_TP_ZOMBIE	25
AP_TP_UNBOUND	-
AP_TP_WRESUMEreq	-
AP_TP_RESUME	-
AP_TP_WRESTARTreq	-
AP_TP_RESTART	-
AP_TP_WRESTART_COMPLETEind	-
AP_TP_HEURISTIC_LOG	-

Table 4-1 Mapping XAP-TP States to OSI TP Service State Numbers

4.1.4 Structure Definitions

The following are definitions for the **ap_tp_cdata_t** and **tp_dialog_env_t** structures.

Note: The **a_assoc_env_t** remains as specified in the **XAP** specification.

The definitions shown here are referenced in subsequent manual pages.

```
/* Dialogue environment structure */

typedef struct {
    int size;              /* buffer size in bytes                        */
    unsigned char *udata;  /* buffer with user encoded                    */
                           /* ATOMIC-ACTION-IDENTIFIER                    */
} ap_aaid_t;

typedef struct {
    int size;              /* buffer size in bytes                        */
    unsigned char *udata;  /* buffer with user encoded BRANCH-IDENTIFIER  */
} ap_brid_t;

typedef struct {
    int size;              /* buffer size in bytes                        */
    unsigned char *udata;  /* buffer with Dialogue Tree Node Identifier   */
} ap_dtnid_t;

typedef struct {
    int size;              /* buffer size in bytes                        */
    unsigned char *udata;  /* buffer transaction tree node identifier     */
} ap_ttnid_t;

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

typedef struct {
    unsigned long mask;
    ap_aaid_t aaid;
    ap_aaid_t next_aaid;
    ap_brid_t brid;
    ap_brid_t next_brid;
    ap_dtnid_t dtnid;
    ap_ttnid_t ttnid;
    ap_ttnid_t next_ttnid;
    ap_apt_t lcl_apt;
    ap_aei_api_id_t lcl_apid;
    ap_aeq_t lcl_aeq;
    ap_aei_api_id_t lcl_aeid;
    ap_tpsu_t lcl_tpsut;
    ap_apt_t rem_apt;
    ap_aei_api_id_t rem_apid;
    ap_aeq_t rem_aeq;
    ap_aei_api_id_t rem_aeid;
    ap_tpsu_t rem_tpsut;
    unsigned long tp_version_sel;
    unsigned long tpfu_sel;
    unsigned long tp_state;
} tp_dialog_env_t;
```

```
typedef struct {
  /* XAP members */
  long              udata_length;     /* length of user-data field          */
  long              rsn;              /* reason for activity/abort/release  */
  long              evt;              /* event that caused abort            */
  long              sync_p_sn;        /* sync point serial number           */
  long              sync_type;        /* sync type                          */
  long              resync_type;      /* resync type                        */
  long              src;              /* source of abort                    */
  long              res;              /* result of primitive                */
  long              res_src;          /* source of result                   */
  long              diag;             /* reason for rejection (if rejected) */
  unsigned long     tokens;           /* tokens identifier                  */
  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-TP members */
  long              tp_options;       /* bit significant TP flags           */
  unsigned long     user_id;          /* abstract syntax or U_ASE identifier*/
  long              tp_fail_count;    /* count of failure conditions        */
  tp_dialog_env_t   *tp_env;          /* dialogue attribute values          */
} ap_tp_cdata_t;
```

The specification of ap_tp_cdata_t in this manner allows compatibility with existing applications using the **ap_cdata_t** definition for access to associations. On some systems this also allows modules compiled with the XAP header file to use the XAP-TP library. It is recommended that applications being upgraded to use the XAP-TP library for concurrent access to dialogues and associations are updated to include the **<xap_tp.h>** header file and to use the **ap_tp_cdata_t** definition where appropriate.

NAME

ap_rcv — receive an XAP-TP primitive

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

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

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

Note: When a single library provides simultaneous access to TP and other ASEs, the function prototype for *ap_rcv*( ) uses *void* * for the cdata pointer, which allows use of the function with the cdata type definitions of each of the ASEs.

Upon return, the value of the **unsigned long** pointed to by *sptype* will indicate the type of primitive that was received. The table below lists the primitives that can be received using *ap_rcv*( ). The following information is provided in the table:

primitive — The symbolic constant defined in **<xap_tp.h>** that is used to identify the primitive.

valid in states — The states during which this primitive may be received.

next state — The state that will be entered upon receipt of this primitive.

States in the following table refer to values of the AP_STATE attribute, except for primitives marked "*", where they represent values of the AP_STATE attribute for instances with the TP_DIALOGUE category selected, and to values of the AP_TP_STATE attribute for instances with *only* the AP_TP_CONTROL category selected.

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
APM_ALLOCATE_CNF (accepted — after APM_ALLOCATE_REQ)	AP_TP_WALLOCcnf	AP_TP_ALLOCATED
APM_ALLOCATE_CNF (accepted — after TP_BEGIN_DIALOGUE_REQ)	AP_TP_WALLOCcnf	AP_TP_DATA_XFER
APM_ALLOCATE_CNF (rejected)	AP_TP_WALLOCcnf	AP_TP_IDLE
APM_ASSOCIATION_LOST_IND	AP_TP_ALLOCATED	AP_TP_IDLE
TP_BEGIN_DIALOGUE_IND (polarised)	AP_TP_IDLE	AP_TP_RECV
TP_BEGIN_DIALOGUE_IND (shared)	AP_TP_IDLE	AP_TP_DATA_XFER
TP_BEGIN_DIALOGUE_CNF (accepted)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind	no state change
TP_BEGIN_DIALOGUE_CNF (rejected, uncoordinated and rollback = FALSE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_WHANDcnf AP_TP_WENDcnf AP_TP_WHAND_GCcnf	AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE
TP_BEGIN_DIALOGUE_CNF (rejected, coordinated and rollback = FALSE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind	AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq
TP_BEGIN_DIALOGUE_CNF (rejected, coordinated and rollback = TRUE)	AP_TP_PREPARING	AP_TP_ROLL_WDONEreq
TP_BEGIN_TRANSACTION_IND	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_WHANDcnf	no state change
TP_DIALOGUE_LOST_IND (Rollback="false") *	AP_TP_LOGGING_READY AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind	AP_TP_LOGGING_READY AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_COMMIT_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
TP_DIALOGUE_LOST_IND (Rollback="true") *	AP_TP_DATAXFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_PREPARING AP_TP_LOGGING_READY AP_TP_WCOMMITind	AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq
TP_COMMIT_IND*	AP_TP_WCOMMITind	AP_TP_COMMIT_WDONEreq
TP_COMMIT_COMPLETE_IND*	AP_TP_WCOMMIT_COMPind	AP_TP_IDLE AP_TP_DATA_XFER AP_TP_RECV AP_TP_ZOMBIE
TP_DATA_IND	AP_TP_DATA_XFER AP_TP_RECV AP_TP_WHANDcnf AP_TP_WENDcnf AP_TP_WREADYind_DATAP	no state change
TP_DEFERRED_END_DIALOGUE_IND	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_WHANDcnf	no state change
TP_DEFERRED_GRANT_CONTROL_IND	AP_TP_RECV AP_TP_ERROR_RECV	no state change
TP_END_DIALOGUE_IND (confirmation = TRUE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_WHANDcnf	AP_TP_WENDrsp AP_TP_WENDrsp AP_TP_DATA_XFER AP_TP_WHANDcnf_WENDrsp
TP_END_DIALOGUE_IND (confirmation = FALSE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_WHANDcnf AP_TP_WENDcnf	AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
TP_END_DIALOGUE_CNF	AP_TP_WENDcnf	AP_TP_IDLE
TP_END_RESUME_IND	AP_TP_RESUME	AP_TP_IDLE
TP_GRANT_CONTROL_IND	AP_TP_RECV AP_TP_ERROR_RECV	AP_TP_DATA_XFER AP_TP_DATA_XFER
TP_HANDSHAKE_IND	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_WHANDcnf AP_TP_WENDcnf	AP_TP_WHANDrsp AP_TP_WHANDrsp AP_TP_DATA_XFER AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDrsp_WENDcnf
TP_HANDSHAKE_CNF	AP_TP_WHANDcnf AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp	AP_TP_DATA_XFER AP_TP_WHANDrsp AP_TP_WENDrsp
TP_HANDSHAKE_AND_GRANT_CONTROL_IND	AP_TP_RECV AP_TP_ERROR_RECV	AP_TP_WHAND_GCrsp AP_TP_DATA_XFER
TP_HANDSHAKE_AND_GRANT_CONTROL_CNF	AP_TP_WHAND_GCcnf	AP_TP_RECV
TP_HEURISTIC_REPORT_IND	AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind	no state change
TP_LOG_DAMAGE_IND *	AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind	no state change
TP_NODE_STATUS_IND*	any	no state change
TP_P_ABORT_IND (coordinated, superior dialogue, awaiting confirm and rollback = FALSE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR AP_TP_WHANDrsp AP_TP_WHAND_GCrsp	AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE
TP_P_ABORT_IND (coordinated not (superior dialogue awaiting confirm) and rollback = FALSE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_LOGGING_READY AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind	AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_LOGGING_READY AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_COMMIT_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
TP_P_ABORT_IND (uncoordinated and rollback = FALSE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDcnf AP_TP_WENDrsp AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp	AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE
TP_P_ABORT_IND (rollback = TRUE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_PREPARING AP_TP_LOGGING_READY	AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq
TP_PREPARE_IND (data_permitted = FALSE)	AP_TP_DATA_XFER AP_TP_RECV	AP_TP_WPREP_ALLreq_DATAP AP_TP_WPREP_ALLreq
TP_PREPARE_IND (data_permitted = TRUE)	AP_TP_RECV	AP_TP_WPREP_ALLreq_DATAP
TP_READY_ALL_IND*	AP_TP_PREPARING	AP_TP_LOGGING_READY
TP_READY_IND	AP_TP_WREADYind AP_TP_WREADYind_DATAP	AP_TP_READY AP_TP_READY
TP_REQUEST_CONTROL_IND	AP_TP_DATA_XFER AP_TP_WHANDcnf	no state change
TP_RESTART_COMPLETE_IND (accepted)	AP_TP_WRESTARTreq AP_TP_WRESTART_COMPLETEind	AP_TP_IDLE AP_TP_IDLE
TP_RESTART_COMPLETE_IND (rejected)	AP_TP_WRESTARTreq AP_TP_WRESTART_COMPLETEind	AP_TP_WRESUMEreq AP_TP_WRESUMEreq

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
TP_ROLLBACK_IND*	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_PREPARING AP_TP_LOGGING_READY AP_TP_WCOMMITind AP_TP_ZOMBIE	AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq
TP_ROLLBACK_COMPLETE_IND*	AP_TP_WROLL_COMPind	AP_TP_IDLE AP_TP_DATA_XFER AP_TP_RECV
TP_U_ABORT_IND (coordinated and rollback = FALSE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind	AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_ZOMBIE AP_TP_COMMIT_WDONEreq AP_TP_COMMIT_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq
TP_U_ABORT_IND (uncoordinated and rollback = FALSE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDcnf AP_TP_WENDrsp AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp	AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
TP_U_ABORT_IND (rollback = TRUE)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_PREPARING AP_TP_LOGGING_READY AP_TP_WCOMMITind	AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq
TP_U_ERROR_IND (shared)	AP_TP_DATA_XFER AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDcnf	AP_TP_DATA_XFER AP_TP_DATA_XFER AP_TP_WHANDrsp AP_TP_WHANDrsp AP_TP_WENDrsp AP_TP_WHANDrsp AP_TP_DATA_XFER
TP_U_ERROR_IND (polarised)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WENDcnf AP_TP_WHAND_GCcnf	AP_TP_ERROR AP_TP_RECV AP_TP_RECV AP_TP_WHANDrsp AP_TP_RECV AP_TP_RECV

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

primitive — The symbolic constant defined in **<xap_tp.h>** that is used to identify the primitive.

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

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

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

Primitive/Attribute Relationships		
Primitive	**Must be Set**	**May Change**
APM_ALLOCATE_CNF	none	AP_DCS AP_DTNID AP_QOS AP_STATE AP_TTNID
APM_ASSOCIATION_LOST_IND	none	AP_STATE
TP_BEGIN_DIALOGUE_IND	AP_BIND_TPADDR AP_LIB_SEL AP_MODE_SEL*	AP_AAID AP_BRID AP_CNTX_NAME AP_DCS AP_DTNID AP_QOS AP_REM_AEID AP_REM_AEQ AP_REM_APID AP_REM_APT AP_REM_PADDR AP_REM_TPSUT AP_ROLE_CURRENT AP_TPFU_SEL AP_TP_SEL AP_STATE
TP_BEGIN_DIALOGUE_CNF	none	AP_STATE
TP_BEGIN_TRANSACTION_IND	none	AP_AAID AP_BRID AP_STATE
TP_COMMIT_IND	none	AP_STATE AP_TP_STATE
TP_COMMIT_COMPLETE_IND	none	AP_AAID AP_BRID AP_NEXT_AAID AP_NEXT_BRID AP_NEXT_TTNID AP_STATE AP_TP_STATE AP_TTNID
TP_DATA_IND	none	none
TP_DEFERRED_END_DIALOGUE_IND	none	none
TP_DEFERRED_GRANT_CONTROL_IND	none	none
TP_DIALOGUE_LOST_IND	none	AP_STATE AP_TP_STATE
TP_END_DIALOGUE_IND	none	AP_STATE
TP_END_DIALOGUE_CNF	none	AP_STATE
TP_END_RESUME_IND	AP_TP_CATEGORY	none
TP_GRANT_CONTROL_IND	none	AP_STATE
TP_HANDSHAKE_IND	none	AP_STATE
TP_HANDSHAKE_CNF	none	AP_STATE
TP_HANDSHAKE_AND_GRANT_CONTROL_IND	none	AP_STATE
TP_HANDSHAKE_AND_GRANT_CONTROL_CNF	none	AP_STATE
TP_HEURISTIC_REPORT_IND	none	AP_STATE

Primitive/Attribute Relationships		
Primitive	**Must be Set**	**May Change**
TP_LOG_DAMAGE_IND	AP_TP_CATEGORY	AP_AAID AP_BRID AP_DTNID AP_TTNID
TP_NODE_STATUS_IND	AP_TP_CATEGORY	AP_STATE AP_TP_STATE
TP_P_ABORT_IND	none	AP_STATE
TP_PREPARE_IND	none	AP_STATE
TP_READY_ALL_IND	none	AP_STATE AP_TP_STATE
TP_READY_IND	none	AP_STATE
TP_REQUEST_CONTROL_IND	none	none
TP_RESTART_COMPLETE_IND	AP_TP_CATEGORY	AP_STATE
TP_ROLLBACK_IND	none	AP_STATE AP_TP_STATE
TP_ROLLBACK_COMPLETE_IND	none	AP_STATE AP_TP_STATE
TP_U_ABORT_IND	none	AP_STATE
TP_U_ERROR_IND	none	AP_STATE

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

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

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

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

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

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

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

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

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

The XAP user is responsible for decoding the user data received in the *ubuf* parameter; see individual manual pages in Chapter 7.

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

AP_ALLOC

If AP_ALLOC is set and the user did not specify an allocation routine on *ap_open*( ) (or *ap_restore*( )) then –1 is returned and the location pointed at by *aperrno_p* is set to the [AP_BAD_FLAGS] error code.

If no space is available in the supplied buffer chain (or the location pointed to by *ubuf* contains NULL), and either AP_ALLOC is not set or AP_ALLOC is set but the user, allocation routine refuses to supply any buffers, then the call to *ap_rcv*( ) fails, –1 is returned and the location pointed to by *aperrno_p* is set to the [AP_NOBUF] error code.

The AP_ALLOC flag setting only takes effect when any supplied buffers have been filled and more data remains to be returned to the user:

— If the AP_ALLOC flag is set, the user allocation routine is called to supply further buffers as they are needed. If the user allocation routine refuses to supply further buffers, then the AP_MORE flag is set and the call to *ap_rcv*( ) completes; 0 is returned. The user must free any buffers allocated by the user allocation routine either by calling the *ap_free*( ) function or by calling the user deallocation routine directly.

— If AP_ALLOC is not set, then the AP_MORE flag is set and 0 is returned.

AP_MORE

This flag is ignored by XAP when *ap_rcv*( ) is called.

Upon return, if all data associated with a primitive has not been received, the AP_MORE bit of the *flags* argument is set, and 0 is returned. This indicates to the user that further *ap_rcv*( ) calls are required to receive the remainder of the data.

If the AP_MORE bit is not set, all data associated with the primitive has been received.

It should be noted that the *sptype* argument must be checked after each invocation of *ap_rcv*( ), since an unsequenced primitive (for example, TP_P_ABORT_IND, TP_ROLLBACK_IND) may arrive before all of the data associated with the first primitive is received. In this case, the remaining data from the original primitive will be lost.

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

If XAP is being used in non-blocking execution mode (AP_NDELAY bit *is* set) and no data is available to be received, *ap_rcv*( ) returns –1 and the location pointed at by *aperrno_p* is set to the [AP_AGAIN] error code.

If rollback is triggered while further fragments of another primitive are being awaited, the rest of the in-progress primitive is lost and *ap_rcv*( ) returns the error code [AP_LOOK] with the AP_MORE bit of the *flags* argument set. This signals the user that XAP-TP has aborted processing of the current primitive. The user should use the *ap_rcv*( ) function to retrieve the pending TP_ROLLBACK_IND primitive. The rest of the in-progress primitive is lost and should not be expected by the user.

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

RETURN VALUE

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

ERRORS

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

[AP_BADF]
Not a presentation service endpoint.

[AP_BADLSTATE]
XAP is in a state where *ap_rcv*( ) is not allowed (for example, AP_UNBOUND).

[AP_BADUBUF]
Either the buffers pointed to by *ubuf* are invalid, or the pointer is NULL and yet AP_ALLOC is not set.

[AP_LOOK]
An event is pending.

[AP_NOBUF]
Out of buffers.

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

[AP_NOMEM]
Out of memory.

[AP_PDUREJ]
XAP rejected the received PDU.

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

NAME

ap_snd — send an XAP-TP primitive

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

This function is used to send a request or response primitive. *fd* identifies the XAP-TP instance for which the primitive is to be sent. The *sptype* parameter identifies which request or response primitive is to be sent. The table below lists the primitives that can be sent using *ap_snd*( ), and the associated states. The following information is provided in the table:

primitive	The symbolic constant defined in **<xap_tp.h>** that is used to identify the primitive.
valid in states	The states during which this primitive may be sent.
next state	The state that will be entered upon successfully issuing this primitive.

States in the following table refer to values of the AP_STATE attribute, except for primitives marked "*" where they represent values of the AP_STATE attribute for instances with the TP_DIALOGUE category selected, and to values of the AP_TP_STATE attribute for instances with *only* the AP_TP_CONTROL category selected.

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
APM_ALLOCATE_REQ	AP_TP_IDLE	AP_TP_WALLOCcnf
A_ABORT_REQ	AP_TP_WALLOCcnf AP_TP_ALLOCATED AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDcnf AP_TP_WENDrsp AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_ROLL_WDONEreq	AP_TP_IDLE AP_TP_IDLE AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDcnf AP_TP_WENDrsp AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_ROLL_WDONEreq
TP_BEGIN_DIALOGUE_REQ	AP_TP_IDLE AP_TP_ALLOCATED	AP_TP_WALLOCcnf AP_TP_DATA_XFER
TP_BEGIN_DIALOGUE_RSP (accepted)	TP_DATA_XFER AP_TP_RECV AP_TP_ERROR AP_TP_WHANDrsp AP_TP_WHAND_GCrsp AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_ROLL_WDONEreq	TP_DATA_XFER AP_TP_RECV AP_TP_ERROR AP_TP_WHANDrsp AP_TP_WHAND_GCrsp AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_ROLL_WDONEreq
TP_BEGIN_DIALOGUE_RSP (rejected)	TP_DATA_XFER AP_TP_RECV AP_TP_ERROR AP_TP_WENDrsp AP_TP_WHANDrsp AP_TP_WHAND_GCrsp AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_ROLL_WDONEreq	AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE
TP_BEGIN_TRANSACTION_REQ	AP_TP_DATA_XFER	AP_TP_DATA_XFER
TP_COMMIT_REQ*	AP_TP_LOGGING_READY	AP_TP_WCOMMITind
TP_DATA_REQ	AP_TP_DATA_XFER AP_TP_WHANDrsp AP_TP_WPREP_ALLreq_DATAP	no state change
TP_DEFERRED_END_DIALOGUE_REQ	AP_TP_DATA_XFER	AP_TP_DATA_XFER
TP_DEFERRED_GRANT_CONTROL_REQ	AP_TP_DATA_XFER	AP_TP_DATA_XFER
TP_DONE_REQ*	AP_TP_COMMIT_WDONEreq AP_TP_ROLL_WDONEreq	AP_TP_WCOMMIT_COMPind AP_TP_WROLL_COMPind
TP_END_DIALOGUE_REQ (confirm=TRUE)	AP_TP_DATA_XFER	AP_TP_WENDcnf
TP_END_DIALOGUE_REQ (confirm=FALSE)	AP_TP_DATA_XFER	AP_TP_IDLE

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
TP_END_DIALOGUE_RSP	AP_TP_WENDrsp	AP_TP_IDLE
TP_FLUSH_REQ	any	no state change
TP_GRANT_CONTROL_REQ	AP_TP_DATA_XFER AP_TP_ERROR	AP_TP_RECV AP_TP_RECV
TP_HANDSHAKE_REQ	AP_TP_DATA_XFER AP_TP_WHANDrsp	AP_TP_WHANDcnf AP_TP_WHANDrsp_WHANDcnf
TP_HANDSHAKE_RSP (shared)	AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDrsp_WENDcnf	AP_TP_DATA_XFER AP_TP_WHANDcnf AP_TP_WENDcnf
TP_HANDSHAKE_RSP (polarised)	AP_TP_WHANDrsp	AP_TP_RECV
TP_HANDSHAKE_AND_GRANT_CONTROL_REQ	AP_TP_DATA_XFER	AP_TP_WHAND_GCcnf
TP_HANDSHAKE_AND_GRANT_CONTROL_RSP	AP_TP_WHAND_GCrsp	AP_TP_DATA_XFER
TP_MANAGE_REQ*	any	no state change
TP_PREPARE_ALL_REQ*	AP_TP_DATA_XFER AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_ZOMBIE	AP_TP_PREPARING AP_TP_PREPARING AP_TP_PREPARING AP_TP_PREPARING AP_TP_PREPARING AP_TP_PREPARING AP_TP_PREPARING
TP_PREPARE_REQ (polarised and data_permitted = FALSE)	AP_TP_DATA_XFER	AP_TP_WREADYind
TP_PREPARE_REQ (shared or data_permitted = TRUE)	AP_TP_DATA_XFER	AP_TP_WREADYind_DATAP
TP_RECOVER_REQ	AP_TP_RESTART	no state change
TP_REQUEST_CONTROL_REQ	AP_TP_RECV AP_TP_WHANDrsp	no state change
TP_RESTART_COMPLETE_REQ	AP_TP_RESTART	AP_TP_WRESTART_COMPLETEind
TP_RESTART_REQ	AP_TP_WRESTARTreq	AP_TP_RESTART
TP_RESUME_REC (success)	AP_TP_WRESUMEreq	AP_TP_RESUME
TP_RESUME_REQ (failure)	AP_TP_WRESUMEreq	AP_TP_WRESTARTreq
TP_ROLLBACK_REQ*	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_PREPARING AP_TP_LOGGING_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_ZOMBIE	AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq

Primitive/State Relationships		
Primitive	**Valid in States**	**Next State**
TP_UPDATE_LOG_DAMAGE_REQ (non-NULL log record)	AP_TP_COMMIT_WDONEreq AP_TP_WCOMMIT_COMPind AP_TP_ROLL_WDONEreq AP_TP_WROLL_COMPind AP_TP_HEURISTIC_LOG	no state change
TP_UPDATE_LOG_DAMAGE_REQ (NULL log record)	AP_TP_HEURISTIC_LOG	node ceases to exist
TP_U_ABORT_REQ (coordinated)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp AP_TP_WREADYind AP_TP_WREADYind_DATAP AP_TP_READY AP_TP_WPREP_ALLreq AP_TP_WPREP_ALLreq_DATAP AP_TP_LOGGING_READY AP_TP_LOGGING_READY AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_ROLL_WDONEreq	AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_ROLL_WDONEreq AP_TP_LOGGING_READY AP_TP_ROLL_WDONEreq AP_TP_WCOMMITind AP_TP_COMMIT_WDONEreq AP_TP_ROLL_WDONEreq
TP_U_ABORT_REQ (not coordinated)	AP_TP_DATA_XFER AP_TP_RECV AP_TP_ERROR_RECV AP_TP_ERROR AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDcnf AP_TP_WENDrsp AP_TP_WHAND_GCcnf AP_TP_WHAND_GCrsp	AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE AP_TP_IDLE
TP_U_ERROR_REQ	AP_TP_DATA_XFER AP_TP_RECV AP_TP_WHANDcnf AP_TP_WHANDrsp AP_TP_WHANDrsp_WHANDcnf AP_TP_WHANDcnf_WENDrsp AP_TP_WHANDrsp_WENDcnf AP_TP_WENDrsp AP_TP_WHAND_GCrsp	AP_TP_DATA_XFER AP_TP_ERROR_RECV AP_TP_WHANDcnf AP_TP_DATA_XFER AP_TP_WHANDcnf AP_TP_WHANDcnf AP_TP_WENDcnf AP_TP_DATA_XFER AP_TP_DATA_XFER

Note: The combination of TP_PREPARE_ALL_REQ, TP_READY_ALL_IND and TP_COMMIT_REQ primitives is the equivalent of the TP_COMMIT request service defined in ISO / IEC 10026-3: 1992 (the OSI TP Protocol).

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

primitive — The symbolic constant defined in **<xap_tp.h>** that is used to identify the primitive.

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

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

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

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

Primitive/Attribute Relationships			
Primitive	**Must be Set**	**May be Used**	**May Change**
APM_ALLOCATE_REQ	AP_BIND_TPADDR AP_CNTX_NAME AP_LIB_SEL AP_MODE_SEL AP_REM_APT	AP_LCL_TPSUT AP_REM_AEID AP_REM_AEQ AP_REM_APID AP_REM_TPSUT AP_ROLE_ALLOWED AP_TPFU_SEL AP_TP_SEL AP_URCH AP_CONTROL_ID	AP_DCS AP_QOS AP_ROLE_CURRENT AP_STATE
A_ABORT_REQ	none	none	AP_STATE
TP_BEGIN_DIALOGUE_REQ	AP_BIND_TPADDR AP_CNTX_NAME AP_LIB_SEL AP_MODE_SEL AP_REM_APT	AP_AAID AP_BRID AP_DTNID AP_LCL_TPSUT AP_REM_AEID AP_REM_AEQ AP_REM_APID AP_REM_TPSUT AP_ROLE_ALLOWED AP_TPFU_SEL AP_TP_SEL AP_TTNID AP_URCH AP_CONTROL_ID	AP_DCS AP_DTNID AP_REM_AEID AP_REM_APID AP_REM_PADDR AP_ROLE_CURRENT AP_STATE AP_TTNID
TP_BEGIN_DIALOGUE_RSP	none	none	AP_STATE
TP_BEGIN_TRANSACTION_REQ	none	AP_AAID AP_BRID	AP_DTNID AP_STATE
TP_COMMIT_REQ	AP_DTNID / AP_TTNID AP_TP_CATEGORY	none	AP_STATE AP_TP_STATE
TP_DATA_REQ	none	none	none
TP_DEFERRED_END_DIALOGUE_REQ	none	none	none
TP_DEFERRED_GRANT_CONTROL_REQ	none	none	none
TP_DONE_REQ	AP_DTNID / AP_TTNID AP_TP_CATEGORY	none	AP_STATE AP_TP_STATE
TP_END_DIALOGUE_REQ	none	none	AP_STATE
TP_END_DIALOGUE_RSP	none	none	AP_STATE
TP_FLUSH_REQ	none	none	none

Primitive/Attribute Relationships			
Primitive	**Must be Set**	**May be Used**	**May Change**
TP_GRANT_CONTROL_REQ	none	none	AP_STATE
TP_HANDSHAKE_REQ	none	none	AP_STATE
TP_HANDSHAKE_RSP	none	none	AP_STATE
TP_HANDSHAKE_AND_GRANT_CONTROL_REQ	none	none	AP_STATE
TP_HANDSHAKE_AND_GRANT_CONTROL_RSP	none	none	AP_STATE
TP_MANAGE_REQ	AP_DTNID/AP_TTNID AP_TP_CATEGORY	none	AP_TP_STATE
TP_PREPARE_ALL_REQ	AP_DTNID/AP_TTNID AP_TP_CATEGORY	none	AP_STATE AP_TP_STATE
TP_PREPARE_REQ	none	none	AP_STATE
TP_RECOVER_REQ	AP_TP_CATEGORY	none	none
TP_REQUEST_CONTROL_REQ	none	none	none
TP_RESUME_REQ	AP_TP_CATEGORY	none	AP_STATE
TP_RESTART_COMPLETE_REQ	AP_TP_CATEGORY	none	AP_STATE
TP_RESTART_REQ	none	none	AP_STATE
TP_ROLLBACK_REQ	AP_DTNID/AP_TTNID AP_TP_CATEGORY	none	AP_STATE AP_TP_STATE
TP_UPDATE_LOG_DAMAGE_REQ	AP_DTNID/AP_TTNID AP_TP_CATEGORY	none	none
TP_U_ABORT_REQ	none	none	AP_STATE
TP_U_ERROR_REQ	none	none	AP_STATE

Note: The use of ‘‘/’’ in the ‘‘Must be Set’’ column indicates ‘‘either/or’’.

ap_snd( ) returns the error code [AP_BADLSTATE] when *sptype* indicates a primitive which is not valid for the current state. For primitives which apply to a transaction node (for example, TP_PREPARE_ALL_REQ, TP_COMMIT_REQ, TP_DONE_REQ), the state check is applied to each of the dialogues of the transaction node, and if it fails then AP_TP_STATE is set to reflect the state of the dialogue causing failure. For other primitives, the state check is against the AP_STATE attribute. An error code of [AP_BADLSTATE] indicates a program logic error. Thus, the XAP-TP user must keep track of the state of the instance, and of the state of the transaction node.

ap_snd( ) returns [AP_LOOK] when the primitive specified by *sptype* is made invalid by an incoming event which has been processed by the underlying service provider but which has not yet been received by the XAP-user.

The [AP_LOOK] error code does not indicate a program logic error. It only indicates that the XAP user should issue *ap_rcv*( ) calls to process one or more outstanding incoming events, and then take action appropriate to the current state of the instance.

The [AP_LOOK] error code can result when any of the following primitives is waiting to be received:

APM_ASSOCIATION_LOST_IND
TP_BEGIN_DIALOGUE_IND
TP_BEGIN_DIALOGUE_CNF (rejected)
TP_DIALOGUE_LOST_IND(Rollback="true")
TP_RESTART_COMPLETE_IND
TP_P_ABORT_IND
TP_ROLLBACK_IND
TP_U_ABORT_IND
TP_U_ERROR_IND

Each of these is disruptive, and terminates any in progress send or receive. An XAP-TP user, after getting [AP_LOOK] and receiving one of these primitives, should assume any send or receive in progress was terminated by XAP-TP.

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

If the primitive being sent is to be accompanied by protocol information, that information must be contained in an **ap_tp_cdata_t** structure pointed to by *cdata*. It is the users responsibility to supply all required information in this structure. Chapter 7 describes the use of the *cdata* argument for each XAP primitive. If no additional protocol information is to be sent with an XAP primitive, cdata may be NULL.

Note: When a single library provides simultaneous access to TP and other ASEs, the function prototype for *ap_snd*( ) uses *void* * for the cdata pointer which allows use of the function with the cdata type definitions of each of the ASEs. When sending a TP primitive the cdata argument must point to an **ap_tp_cdata_t** structure as described above.

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

The **ap_osi_vbuf_t** structure is defined as shown below.

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

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

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

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

The API user is responsible for encoding the user data passed to XAP in the *ubuf* parameter; see individual manual pages in Chapter 7.

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

AP_MORE
This flag indicates that data associated with the specified primitive will be sent with multiple *ap_snd*( ) calls. Each *ap_snd*( ) call with the AP_MORE bit set indicates that another *ap_snd*( ) will follow with additional data associated with the specified primitive.

The value of the *sptype* argument must be the same for all *ap_snd*( ) calls used to send a single primitive.

Calling *ap_snd*( ) with the AP_MORE bit reset signals that the primitive is complete.

AP_QUERY_DATA_PENDING
This flag indicates to XAP that a check shall be made for the availability of incoming data on the connection, and that if data is available this shall be indicated by returning a result of –1 and setting the location pointed at by *aperrno_p* as indicated below.

AP_FLUSH
This flag indicates to XAP that the primitive being passed is to end a concatenation sequence.

This flag may not be set in conjunction with the AP_MORE flag.

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

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

1. the user resets the AP_MORE flag

   and

2. XAP returns success; that is, does not return [AP_AGAIN]

   or

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

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

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

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

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

RETURN VALUE

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

ERRORS

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

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

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

[AP_BADATTRVAL]
An environment attribute override contains an invalid value for the attribute.

[AP_BADDATA]
User data not allowed on this service. User data and presentation pci exceeds 512 bytes.

[AP_BADENV]
A mandatory environment attribute is not set.

[AP_BADFLAGS]
The specified combination of flags is invalid.

[AP_BADF]
Not a presentation service endpoint.

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

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

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

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

[AP_LOOK]
A pending event requires attention.

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

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

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

Chapter 5

XAP-TP Commands

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

NAME

xap_tp_osic — XAP-TP Library OSI information compiler.

SYNOPSIS

```
xap_tp_osic [ options ] files
```

DESCRIPTION

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

The format of the **ap_env_file** input files is defined in Chapter 3 on page 73.

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

-o outfile: write output to **outfile** instead of **ap_osi.env**.

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

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

```
AP_LIB_SEL, AP_ROLE_ALLOWED, AP_CNTX_NAME,
AP_MODE_SEL, AP_FLAGS, AP_TP_SEL, AP_TP_CATEGORY,
AP_TPFU_SEL, AP_CONTROL_ID, AP_BIND_TPADDR,
AP_AAID, AP_NEXT_AAID, AP_BRID, AP_NEXT_BRID,
AP_URCH, AP_TTNID, AP_NEXT_TTNID, AP_LCL_TPSUT,
AP_REM_APT, AP_REM_APID, AP_REM_AEQ, AP_REM_AEID,
AP_REM_TPSUT, AP_TP_COPYENV.
```

FILES

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

CAVEAT

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

DIAGNOSTICS

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

Cannot open file
: One of the specified input files cannot be opened for reading.

Syntax error
: There is a syntax error in the **ap_env_file** file. Refer to the **ap_env_file** manual page for a description of the proper syntax.

Illegal attribute
: An illegal attribute name was specified in the **ap_env_file** file.

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

Duplicate attribute ignored
: More than one assignment was encountered for a single attribute. The first value is used.

Chapter 6

XAP-TP File Formats

This chapter defines the format of files used by XAP-TP.

6.1 Environment File

This defines the format of an XAP-TP Library initialisation file.

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

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

- Assignment pairs of the form:

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

  where `<attribute name>` is the name of an XAP-TP library environment attribute (see Chapter 3 on page 73) and `<value>` is a legal value for the attribute.

- C-style comments (`/*....*/`) with the syntax and semantics defined by ISO C.
- `#include` lines with the syntax and semantics defined by the ISO C preprocessor.
- `#define` lines with the syntax and semantics defined by the ISO C preprocessor for the `#define identifier token-sequence` form.

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

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

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

attribute name	type	value format
AP_AAID	ap_aaid_t	encoded string
AP_BRID	ap_brid_t	encoded string
AP_BIND_TPADDR	ap_tpaddr_t	address
AP_CNTX_NAME	ap_objid_t	object identifier
AP_CONTROL_ID	ap_cid_t	octet string
AP_FLAGS	unsigned long	bitmask
AP_LCL_TPSUT	ap_tpsut_t	encoded string
AP_LIB_SEL	long	integer constant
AP_MODE_SEL	long	integer constant
AP_NEXT_AAID	ap_aaid_t	encoded string
AP_NEXT_BRID	ap_brid_t	encoded string
AP_NEXT_TTNID	ap_ttnid_t	octet string
AP_REM_AEID	ap_aei_api_id_t	encoded string
AP_REM_AEQ	ap_aeq_t	encoded string
AP_REM_APID	ap_aei_api_id_t	encoded string
AP_REM_APT	ap_apt_t	encoded string
AP_REM_TPSUT	ap_tpsut_t	encoded string
AP_TTNID	ap_ttnid_t	octet string
AP_URCH	ap_urch_t	octet string
AP_ROLE_ALLOWED	unsigned long	bitmask
AP_TP_CATEGORY	long	integer constant
AP_TP_COPYENV	long	integer constant
AP_TP_SEL	long	integer constant
AP_TPFU_SEL	long	bitmask

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

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

Address

Values in this format must be given as

```
{[apt], [apid], [aeq], [aeid], count [, {tpsut [, tpsut]*}]}
```

where apt, apid, aeq, aeid, count and tpsut are defined as follows:

apt: A value in the encoded string format of any length.

apid: A value in the encoded string format of any length.

aeq: A value in the encoded string format of any length.

aeid: A value in the encoded string format of any length.

count: An integer constant value which specifies the number of tpsut values that follow. A value of zero is allowed and implies that no list of tpsut values follows.

tpsut: One or more tpsut values may be specified here in a comma separated list. Each tpsut is represented by a value in the encoded string format.

Example:

```
AP_BIND_TPADDR = { {06074150546C6F6F70}, {020101}, {020101}, {020101}, \
                 1, { {13056C65616631} } }
```

Bitmask

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

Example:

```
AP_TPFU_SEL = AP_TP_POLARIZED_CONTROL  | AP_TP_COMMIT_AND_UNCHAINED
```

Integer Constant

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

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

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

Example:

```
AP_ROLE_ALLOWED = AP_RESPONDER
```

Object Identifier

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

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

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

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

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

Examples:

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

Octet String

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

Examples:

```
AP_CONTROL_ID = {000ff0ff}
AP_TTNID = {"tran 1259"}
```

Encoded String

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

Example:

```
AP_REM_APT = {06062B80CE0F0107}
```

Chapter 7

XAP-TP Primitives

This chapter presents manual page definitions for each of the primitives of the underlying OSI services to which the XAP provides access via the *ap_snd*( ) and *ap_rcv*( ) functions. Each manual page provides a short description of an OSI TP, ACSE or Presentation Layer primitive, including the circumstances under which it may be sent or received, and a detailed description of the parameters associated with it.

NAME

APM_ALLOCATE_REQ — used to request allocation of an association for an outgoing dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The APM_ALLOCATE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request allocation of an association prior to initiating an outgoing dialogue with a TP_BEGIN_DIALOGUE_REQ primitive. If the APM_ALLOCATE_REQ primitive is accepted, no further primitives can be issued, except an A_ABORT_REQ, until an APM_ALLOCATE_CNF primitive is received.

The result of the allocation attempt will be reported by receipt of an APM_ALLOCATE_CNF primitive.

The primitive is only used when automatic allocation is not being used on the TP_BEGIN_DIALOGUE_REQ primitive.

An association allocated by this primitive will be suitable for use with the TP functional units requested. Where an application context supports both provider and application supported transactions, an attempt to allocate an association for use with the commit functional unit may fail where an attempt to allocate without would be successful.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to APM_ALLOCATE_REQ.

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

```
long tp_options;         /* synchronous allocation?        */
a_assoc_env_t *env;      /* association attribute values   */
tp_dialog_env_t *tp_env; /* dialogue attribute values      */
```

cdata→tp_options can be set to indicate whether the primitive is to return an error if an association is not immediately available from a suitable pool or is to wait for establishment of a new association. The possible bit settings of *cdata→tp_options* for this primitive are:

AP_TP_SYNC_ALLOC
: If set, indicates that an APM_ALLOCATE_CNF primitive rejecting the allocation attempt is to be issued immediately if an association cannot be allocated from a suitable pool.

If unset, indicates that the allocation attempt is to remain pending until a association is allocated from a suitable pool.

AP_TP_CONT_WINNER
If set, indicates that the association must be a contention-winner.

cdata→env can be used to override XAP environment attribute values used by this primitive. If no attribute values are to be overridden, *cdata→env* may be set to NULL. Otherwise, *cdata→env* must point to an **a_assoc_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask        */
objid_t cntx_name;           /* AP_CNTX_NAME    */
ap_qos_t qos;                /* AP_QOS          */
```

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

Flag	Parameter	Field
AP_CNTX_NAME_BIT	Application Context Name	*cntx_name*
AP_QOS_BIT	Quality of Service	*qos*

cdata→tp_env can be used to override XAP environment attribute values used by this primitive. If no attribute values are to be overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;            /* bit mask        */
ap_apt_t rem_apt;              /* AP_REM_APT      */
ap_aei_api_id_t rem_apid;      /* AP_REM_APID     */
ap_aeq_t rem_aeq;              /* AP_REM_AEQ      */
ap_aei_api_id_t rem_aeid;      /* AP_REM_AEID     */
unsigned long tp_version_sel;  /* AP_TP_SEL       */
unsigned long tpfu_sel;        /* AP_TPFU_SEL     */
```

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

Flag	Parameter	Field
AP_REM_APT_BIT	Remote Application Process Title	*rem_apt*
AP_REM_APID_BIT	Remote Application Process Invocation Identifier	*rem_apid*
AP_REM_AEQ_BIT	Remote Application Entity Qualifier	*rem_aeq*
AP_REM_AEID_BIT	Remote Application Entity Invocation Identifier	*rem_aeid*
AP_TP_SEL_BIT	TP Version Selector	*tp_version_sel*
AP_TPFU_SEL_BIT	TP Requirements	*tpfu_sel*

ubuf Not used.

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

RETURN VALUE

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

ERRORS

In addition to the errors listed on the *ap_snd*( ) manual page, the following APM_ALLOCATE_REQ errors may occur:

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

[AP_TP_BADCD_TP_OPTIONS]
The setting of *cdata→tp_options* is invalid.

NAME

APM_ALLOCATE_CNF — used to confirm an association allocation request

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The APM_ALLOCATE_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm allocation of an association to support an outgoing begin dialogue, or to inform the application of failure to allocate or establish an association.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long res;               /* result of request        */
long res_src;           /* source of result         */
long diag;              /* reason (if rejected)     */
```

cdata→res will be set to indicate the result of the association allocation request. The possible values for *cdata→res* are as follows:

AP_ACCEPT
: Association has been allocated.

AP_REJ_PERM
: Association allocation or establishment permanently rejected.

AP_REJ_TRAN
: Association allocation or establishment temporarily rejected.

The argument *cdata→res_src* indicates the source of the result and will be one of the following:

AP_APM_SERV_PROV
: APM service provider (Association Pool Manager)

AP_TP_SERV_PROV
: TP service provider.

AP_ACSE_SERV_USER
: ACSE service user.

AP_ACSE_SERV_PROV
ACSE service provider.

AP_PRES_SERV_PROV
Presentation service provider.

AP_SESS_SERV_PROV
Session service provider.

AP_TRAN_SERV_PROV
Transport service provider.

The value of the *cdata→diag* is not meaningful when the value of *cdata→res* is AP_ACCEPT. When the value is either AP_REJ_PERM or AP_REJ_TRAN, the possible values for *cdata→diag* depend on the source of the result. If the source of the result is AP_TP_SERV_PROV, *cdata→diag* will be either the value:

AP_TP_NRSN
No reason given.

or a bit significant field value giving the reason(s) for rejection. The bits which may be set are:

AP_TP_CCR_V2_NAVAIL
CCR version 2 is not available.

AP_TP_VER_NAVAIL
TP version incompatibility.

AP_TP_CW_REJ
Contention winner assignment rejected.

AP_TP_BM_REJ
Bid mandatory value rejected.

If the source of the result is AP_APM_SERV_PROV, *cdata→diag* will be one of the following:

AP_TP_ASSOC_NVAIL
Synchronous allocation from pool was requested via the AP_TP_SYNC_ALLOC bit setting in *cdata→tp_options*, but no suitable association could be found. Pool limits permit establishment of further associations so the XAP-TP provider will have commenced establishment of another association for the pool prior to returning.

AP_TP_BAD_AET
The specified local or remote AET is unknown.

AP_TP_BAD_POOL
No usable pool definition is available for associations joining the specified AETs.

AP_TP_POOL_LIMIT
Synchronous allocation from pool was requested via the AP_TP_SYNC_ALLOC bit setting in *cdata→tp_options*, but pool limits prevent further associations from being established.

AP_TP_POOL_TIMEOUT
The pool manager has exceeded its configured maximum waiting time for an association to be freed back into the pool.

AP_TP_DIALOGUE_REFUSED
The pool manager cannot join a transaction mode dialogue to a transaction node which is in the termination phase.

When the value of *cdata→res_src* is one of AP_ACSE_SERV_USER, AP_ACSE_SERV_PROV, AP_PRES_SERV_PROV, AP_SESS_SERV_PROV or AP_TRAN_SERV_PROV, a failure occurred in the attempt to establish an association, and the value of the *cdata→diag* argument will be set to one of the values detailed on the A_ASSOC_CNF manual page of the **XAP** specification.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

APM_ASSOCIATION_LOST_IND — used to indicate loss of an allocated association

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The APM_ASSOCIATION_LOST_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that an association allocated by an APM_ALLOCATE_REQ primitive has been lost.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long rsn;       /* reason for association loss   */
long evt;       /* event that caused loss        */
long src;       /* source of loss                */
```

cdata→src indicates the source of the association loss. Possible values for this field are as follows:

AP_APM_SERV_PROV
: APM service provider (Association Pool Manager).

AP_ACSE_SERV_USER
: ACSE service user.

AP_ACSE_SERV_PROV
: ACSE service provider.

AP_PRES_SERV_PROV
: Presentation service provider.

AP_SESS_SERV_PROV
: Session service provider.

AP_TRAN_SERV_PROV
: Transport service provider.

cdata→rsn indicates the reason for the association loss. The possible values for *cdata→rsn* depend on the value of *cdata→src*. If *cdata→src* is set to AP_APM_SERV_PROV, *cdata→rsn* will be set to the following:

AP_TP_IN_DIALOGUE
An incoming dialogue has claimed the association.

If *cdata→src* is set to AP_ACSE_SERV_USER, the association loss was caused by either an A_ABORT_IND or an A_RELEASE_IND primitive being received. *cdata→rsn* will be set to one of the following:

AP_TP_ABORTED
An A_ABORT_IND was received.

AP_REL_NORMAL
Normal release request.

AP_REL_URGENT
Urgent release request.

AP_REL_USER_DEF
User defined release request.

AP_RSN_NOVAL
A_RELEASE_IND received but reason not specified.

If *cdata→src* is set to AP_ACSE_SERV_PROV, the association loss was caused by an A_ABORT_IND being received. *cdata→rsn* will be set to the following:

AP_TP_ABORTED
An A_ABORT_IND was received.

If *cdata→src* is set to one of the values AP_PRES_SERV_PROV, AP_SESS_SERV_PROV or AP_TRAN_SERV_PROV, the association loss was caused by an A_PABORT_IND primitive being received, the values of *cdata→rsn* and *cdata→evt* will be set at detailed on the A_PABORT_IND manual page of the **XAP** specification.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_BEGIN_DIALOGUE_REQ — used to begin a dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_BEGIN_DIALOGUE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to begin a dialogue between two application entities.

If the underlying association has not been previously allocated (AP_TP_ASSOC_ALLOCATED bit not set in *cdata→tp_options*) then the XAP-TP provider queues the request internally, performs an implicit APM_ALLOCATE_REQ, and reports the result to the user by issuing an APM_ALLOCATE_CNF primitive. If the APM_ALLOCATE_CNF reports success, the TP_BEGIN_DIALOGUE_REQ has been submitted, otherwise it has been discarded, and no association will have been allocated. If the user wishes to abandon a TP_BEGIN_DIALOGUE_REQ whilst association allocation is being performed, an A_ABORT_REQ can be issued.

The caller can include the dialogue into a particular node of a dialogue tree by providing either the AP_DTNID or the AP_TTNID (if one has been associated) environment attribute. (See also Section 2.6 on page 35 and **Using a Local Identifier** on page 37.)

Note that the AP_LCL_APT, AP_LCL_APID, AP_LCL_AEQ, AP_LCL_AEID and AP_LCL_TPSUT attributes of all dialogues belonging to a node in the tree must be the same.

If adding a new branch to the transaction tree, the caller may give the BrId in the AP_BRID environment attribute. If omitted, XAP-TP will allocate a Branch Identifier. If the caller supplies a value in AP_AAID, it must be the same as the AAId currently in use for the TPSUIs transaction tree.

When starting a new transaction tree, the user may set the AP_AAID and AP_BRID singly or in combination. XAP-TP will allocate the AAId and/or the BrId if they are not supplied.

Note that when the user allocates an AAId and/or BRId, they must conform to the OSI TP standard. XAP-TP validates the format of user-supplied AAIds and BrIds, and that a user-supplied BrId contains the AET the instance is bound to.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_BEGIN_DIALOGUE_REQ.

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

```
long udata_length;       /* length of user-information field */
long tp_options;         /* allocate/start trans/confirm?    */
a_assoc_env_t *env;      /* association attribute values     */
tp_dialog_env_t *tp_env; /* dialogue attribute values        */
```

cdata→udata_length may be set to the total number of octets of encoded user-information that will be sent with this primitive. If the amount of data to be sent with this primitive is not known, this field should be set to –1. In some XAP-TP implementations, setting this field may improve performance.

cdata→tp_options will be set to indicate if an association has already been allocated for the dialogue, if the dialogue is to be started in transaction mode, and if confirmation is required. The possible bit settings of *cdata→tp_options* for this primitive are:

AP_TP_ASSOC_ALLOCATED
: Indicates that a prior call to APM_ALLOCATE_REQ has allocated the association for the begin dialogue. If unset, an association will be allocated from a suitable pool.

AP_TP_SYNC_ALLOC
: If set, indicates that an APM_ALLOCATE_CNF primitive rejecting the allocation attempt is to be issued immediately if an association cannot be allocated from a suitable pool.

  The error code returned depends on the status of the pool; if unset, indicates that the allocation attempt is to remain pending until a association is allocated from a suitable pool.

AP_TP_CONT_WINNER
: If set, indicates that the association must be a contention-winner.

AP_TP_TRANSACTION
: When the unchained transactions functional unit is selected, it indicates that the dialogue is to be started in transaction mode.

AP_TP_CONFIRM
: Explicit confirmation by the remote TPSUI is required.

Notes:

1. When the chained transactions functional unit is selected, a dialogue always starts in transaction mode.

2. When joining a transaction mode dialogue to an existing transaction node, on a specifically allocated association, if the transaction node is in termination, the TP_BEGIN_DIALOGUE_REQ primitive will fail [AP_BADLSTATE] because the Node is in a bad state to accept the primitive. When using automatic association allocation, this situation is not detected until the association has been acquired, and so the TP_BEGIN_DIALOGUE_REQ is accepted, but the result AP_TP_DIALOGUE_REFUSED is reported on the APM_ALLOCATE_CNF primitive.

cdata→env can be used to override XAP environment attribute values used by this primitive. If no attribute values are to be overridden, *cdata→env* may be set to NULL. Otherwise, *cdata→env* must point to an **a_assoc_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask        */
objid_t cntx_name;           /* AP_CNTX_NAME    */
ap_qos_t qos;                /* AP_QOS          */
```

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

Flag	Parameter	Field
AP_CNTX_NAME_BIT	Application Context Name	*cntx_name*
AP_QOS_BIT	Quality of Service	*qos*

cdata→tp_env can be used to override XAP environment attributes values used as parameters to the TP-BEGIN-DIALOGUE request service. If no attribute values are to be overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;            /* bit mask        */
ap_aaid_t aaid;                /* AP_AAID         */
ap_brid_t brid;                /* AP_BRID         */
ap_dtnid_t dtnid;              /* AP_DTNID        */
ap_ttnid_t ttnid;              /* AP_TTNID        */
ap_tpsu_t lcl_tpsut;           /* AP_LCL_TPSUT    */
ap_apt_t rem_apt;              /* AP_REM_APT      */
ap_aei_api_id_t rem_apid;      /* AP_REM_APID     */
ap_aeq_t rem_aeq;              /* AP_REM_AEQ      */
ap_aei_api_id_t rem_aeid;      /* AP_REM_AEID     */
ap_tpsu_t rem_tpsut;           /* AP_REM_TPSUT    */
unsigned long tp_version_sel;  /* AP_TP_SEL       */
unsigned long tpfu_sel;        /* AP_TPFU_SEL     */
```

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

Flag	Parameter	Field
AP_AAID_BIT	Atomic Action Identifier	*aaid*
AP_BRID_BIT	Branch Identifier	*brid*
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*
AP_LCL_TPSUT_BIT	Local TP Service User Title	*lcl_tpsut*
AP_REM_APT_BIT	Remote Application Process Title	*rem_apt*
AP_REM_APID_BIT	Remote Application Process Invocation Identifier	*rem_apid*
AP_REM_AEQ_BIT	Remote Application Entity Qualifier	*rem_aeq*
AP_REM_AEID_BIT	Remote Application Entity Invocation Identifier	*rem_aeid*
AP_REM_TPSUT_BIT	Remote TP Service User Title	*rem_tpsut*
AP_TP_SEL_BIT	TP Version Selector	*tp_version_sel*
AP_TPFU_SEL_BIT	TP Requirements	*tpfu_sel*

ubuf Use of *ubuf* is described on the *ap_snd*( ) manual page. Data carried in the *ubuf* buffer(s) must be encoded according to the definition specified in ISO / IEC 10026-3: 1992 (the OSI TP Protocol), ([30] IMPLICIT SEQUENCE OF EXTERNAL).

Note: It is only permissible to include user data on the begin dialogue if the association has been previously allocated and the AP_TP_ASSOC_ALLOCATED bit is set in *cdata→tp_options*.

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

RETURN VALUE

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

ERRORS

In addition to the errors listed on the *ap_snd*( ) manual page, the following TP_BEGIN_DIALOGUE_REQ errors may occur:

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

[AP_TP_BAD_UDATA]
User data has been passed on the primitive and the AP_TP_ASSOC_ALLOCATED bit is not set in *cdata→tp_options*.

[AP_TP_BADCD_TP_OPTIONS]
The *cdata→tp_options* setting is invalid.

NAME

TP_BEGIN_DIALOGUE_IND — used to indicate a request to establish a dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

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

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long tp_options;          /* start transaction/confirm? */
tp_dialog_env_t *tp_env; /* dialogue attribute values  */
```

cdata→tp_options will be set to indicate if the dialogue is being started in transaction mode, and if confirmation is required. The possible values for *cdata→tp_options* are as follows:

AP_TP_TRANSACTION
: When the unchained transactions functional unit is selected, indicates that the dialogue is in transaction mode.

AP_TP_CONFIRM
: This TP_BEGIN_DIALOGUE_IND requires explicit confirmation by a TP_BEGIN_DIALOGUE_RSP primitive if accepted by the user.

Note that if the chained transactions functional unit is selected, then the dialogue is always started in transaction mode.

If the AP_TP_COPYENV attribute in the XAP environment is FALSE, the values corresponding to the parameters of the TP-BEGIN-DIALOGUE indication service will not be returned in the *cdata* argument and *cdata→tp_env* will be set to NULL. If AP_TP_COPYENV is TRUE, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask       */
ap_aaid_t aaid;              /* AP_AAID        */
ap_brid_t brid;              /* AP_BRID        */
```

```
ap_dtnid_t dtnid;              /* AP_DTNID        */
ap_tpsu_t lcl_tpsut;           /* AP_LCL_TPSUT    */
ap_apt_t rem_apt;              /* AP_REM_APT      */
ap_aei_api_id_t rem_apid;      /* AP_REM_APID     */
ap_aeq_t rem_aeq;              /* AP_REM_AEQ      */
ap_aei_api_id_t rem_aeid;      /* AP_REM_AEID     */
ap_tpsu_t rem_tpsut;           /* AP_REM_TPSUT    */
unsigned long tp_version_sel;  /* AP_TP_SEL       */
unsigned long tpfu_sel;        /* AP_TPFU_SEL     */
```

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

Flag	Parameter	Field
AP_AAID_BIT	Atomic Action Identifier	*aaid*
AP_BRID_BIT	Branch Identifier	*brid*
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_LCL_TPSUT_BIT	Local TP Service User Title	*lcl_tpsut*
AP_REM_APT_BIT	Remote Application Process Title	*rem_apt*
AP_REM_APID_BIT	Remote Application Process Invocation Identifier	*rem_apid*
AP_REM_AEQ_BIT	Remote Application Entity Qualifier	*rem_aeq*
AP_REM_AEID_BIT	Remote Application Entity Invocation Identifier	*rem_aeid*
AP_REM_TPSUT_BIT	Remote TP Service User Title	*rem_tpsut*
AP_TP_SEL_BIT	TP Version Selector	*tp_version_sel*
AP_TPFU_SEL_BIT	TP Requirements	*tpfu_sel*

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

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

RETURN VALUE

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

ERRORS

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

NAME

TP_BEGIN_DIALOGUE_RSP — used to respond to a begin dialogue indication

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_BEGIN_DIALOGUE_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to a begin dialogue request. This primitive may always be issued in response to a TP_BEGIN_DIALOGUE_IND to reject the dialogue, but may only be issued to confirm acceptance of the dialogue when confirmation has been specifically requested via the AP_TP_CONFIRM bit setting of *cdata→tp_options* on the TP-BEGIN-DIALOGUE indication.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_BEGIN_DIALOGUE_RSP.

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

```
long udata_length;   /* length of user-information field */
long res;            /* result of request                */
```

cdata→udata_length may be set to the total number of octets of encoded user-information that will be sent with this primitive. If the amount of data to be sent with this primitive is not known, this field should be set to –1. In some XAP-TP implementations, setting this field may improve performance.

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

AP_TP_ACCEPT
: Accept the dialogue.

AP_TP_REJ_USER
: Dialogue rejected.

ubuf Use of the *ubuf* argument is described on the *ap_snd*( ) manual page. Data carried in the *ubuf* buffer(s) must be encoded according to the definition specified in ISO / IEC 10026-3: 1992 (the OSI TP Protocol) ([30] IMPLICIT SEQUENCE OF EXTERNAL).

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

RETURN VALUE

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

ERRORS

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

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

NAME

TP_BEGIN_DIALOGUE_CNF — used to confirm a begin dialogue request

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_BEGIN_DIALOGUE_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm establishment of a dialogue between two application entities, or to inform the application of the failure to establish the dialogue.

Note that a TP_BEGIN_DIALOGUE_CNF indicating acceptance will only be returned if confirmation was requested in the initiating TP_BEGIN_DIALOGUE_REQ (AP_TP_CONFIRM bit of *cdata→tp_options* was set).

If a dialogue has been rejected the *cdata→tp_options* will indicate whether the transaction is being rolled back as a result via the AP_TP_ROLLBACK bit setting.

If the dialogue has been rejected, and the transaction is being rolled back as a result or the transaction was already being rolled back (state prior to receiving the primitive was AP_TP_ROLL_WDONEreq or AP_TP_WROLL_COMPind), then the current failure count returned in *cdata→tp_fail_count* will have been incremented, and the failure is propagated to each other active coordinated instance of the node and to the control instance (if it is not an active coordinated instance) as a TP_DIALOGUE_LOST_IND primitive. This propagation allows the user to coordinate failure-related actions.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long res;                /* result of request           */
long diag;               /* reason (if rejected)        */
long tp_options;         /* rollback transaction?       */
long tp_fail_count;      /* count of failure conditions */
tp_dialog_env_t *tp_env; /* dialogue attribute values   */
```

cdata→res will be set to indicate the result of the dialogue request. The possible values for *cdata→res* are as follows:

AP_TP_ACCEPT
Accept the dialogue.

AP_TP_REJ_PROV
Dialogue rejected by TP service provider.

AP_TP_REJ_USER
Dialogue rejected by TP service user.

If the dialogue has been rejected and the value of *cdata→res* is AP_TP_REJ_PROV, then the value of *cdata→diag* indicates the cause of the rejection. *cdata→diag* will be one of the following:

AP_TP_NRSN
No reason given.

AP_TP_RECIPIENT_UNKNOWN
The parameters identifying the recipient application-entity-invocation do not identify a known application-entity-invocation.

AP_TP_TPSUT_UNKNOWN
The called TPSUT was not recognised.

AP_TP_TPSUT_NVAIL_PERM
The called TPSUT was recognised, but is permanently unavailable.

AP_TP_TPSUT_NVAIL_TRAN
The called TPSUT was recognised, but is temporarily unavailable.

AP_TP_TPSUT_NEEDED
Called TPSUT was not provided in TP_BEGIN_DIALOGUE_REQ and is required.

AP_TP_FU_NSUP
One or more of the functional units selected in the TP_BEGIN_DIALOGUE_REQ are not supported by the recipient for the dialogue.

AP_TP_FU_COMB_NSUP
The combination of functional units selected in the TP_BEGIN_DIALOGUE_REQ are not supported by the recipient for the dialogue.

AP_TP_ASSOC_RES
The association is reserved for usage by the remote system.

When the value of *cdata→res* is *not* AP_TP_ACCEPT, *cdata→tp_options* indicates if the transaction in which the recipient is involved is to be rolled back. The bit value in *cdata→tp_options* used is:

AP_TP_ROLLBACK
If set, the transaction is to be rolled back. If unset, no rollback is to occur.

cdata→tp_fail_count holds the number of failure conditions which have occurred on the transaction node since the start of the transaction, and must be used when issuing a TP_DONE_REQ primitive to acknowledge completion of any failure related actions. This is only valid if the dialogue was coordinated.

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-BEGIN-DIALOGUE confirmation service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in cdata and *cdata→tp_env* is set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* points to a **tp_dialog_env_t**

structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask  */
unsigned long tpfu_sel       /* AP_TPFU   */
```

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

Flag	Parameter	Field
AP_TPFU_SEL_BIT	TP Requirements	*tpfu_sel*

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

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

RETURN VALUE

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

ERRORS

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

NAME

TP_BEGIN_TRANSACTION_REQ — used to commence a transaction on a dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_BEGIN_TRANSACTION_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to commence a transaction on a dialogue.

When adding a new branch to the transaction tree the caller may give the BrId in the AP_BRID environment attribute. If omitted, XAP-TP will allocate a Branch Identifier. If the caller supplies a value in AP_AAID, it must be the same as the AAId currently in use for the TPSUIs transaction tree.

When starting a new transaction tree, the user may set the AP_AAID and AP_BRID singly or in combination. XAP-TP will allocate the AAId and/or the BrId if they are not supplied.

Note that when the user allocates an AAId and/or BRId, they must conform to the OSI TP standard. XAP-TP validates the format of user-supplied AAIds and BrIds, and that a user-supplied BrId contains the AET the instance is bound to.

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

Note: An attempt to begin a transaction branch on a dialogue which is part of a transaction node which is in the termination phase (undergoing commitment or rollback) will be rejected with the error code [AP_BADLSTATE], because the Node is in a bad state to accept the primitive.

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_BEGIN_TRANSACTION_REQ.

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values */
```

cdata→tp_env can be used to override XAP environment attributes values used as parameters to the TP-BEGIN-DIALOGUE request service. If no attribute values are to be overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_aaid_t aaid;              /* AP_AAID    */
ap_brid_t brid;              /* AP_BRID    */
```

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

Flag	Parameter	Field
AP_AAID_BIT	Atomic Action Identifier	*aaid*
AP_BRID_BIT	Branch Identifier	*brid*

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_BEGIN_TRANSACTION_IND — is used to indicate a request to commence a transaction

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_BEGIN_TRANSACTION_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a request to commence a transaction on the dialogue.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-BEGIN-TRANSACTION indication service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in *cdata* and *cdata→tp_env* will be set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_aaid_t aaid;              /* AP_AAID    */
ap_brid_t brid;              /* AP_BRID    */
```

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

Flag	Parameter	Field
AP_AAID_BIT	Atomic Action Identifier	*aaid*
AP_BRID_BIT	Branch Identifier	*brid*

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_COMMIT_REQ — used to request commitment of a transaction

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_COMMIT_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request commitment of the transaction identified by *dtnid* or *ttnid*.

For a superior it indicates the transaction is to be committed, otherwise (for an intermediate or leaf node) indicates the first phase of commitment has been completed and the caller is ready to accept commitment or rollback of the transaction.

The primitive must be issued on the nominated control instance for the transaction node.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_COMMIT_REQ.

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to override XAP environment attributes values used as parameters to the TP-COMMIT request service. If no attribute values are to be overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The user may identify the transaction to be committed using one of:

- the Dialogue Tree Node Identifier AP_DTNID
- the Transaction Tree Node Identifier AP_TTNID.

If the XAP instance has the TP_DIALOGUE category selected, it is only possible to request commitment of a transaction on the dialogue tree referenced by a dialogue on the XAP instance.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

[AP_TP_BAD_NODE]
An extant transaction node is not identified by AP_DTNID or AP_TTNID.

NAME

TP_COMMIT_IND — used to indicate a request to commit a transaction

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_COMMIT_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a request for the transaction manager to commit the transaction identified by *dtnid* or *ttnid*. This primitive is issued on the nominated control instance for the transaction node, and on each of the active coordinated instances of the node (except when it is also the nominated control instance).

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-COMMIT indication service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in *cdata* and *cdata→tp_env* will be set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The dialogue tree node identifier *dtnid* will be set to the value allocated to the TPSUI when it was started. The transaction tree node identifier *ttnid* will only be present if the user has set a value in this attribute for the TPSUI.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_COMMIT_COMPLETE_IND — used to indicate completion of a transaction

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_COMMIT_COMPLETE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the transaction identified by *dtnid* or *ttnid* has been completed.

This primitive is passed to the user on each coordinated instance of the node, and also on the nominated control instance for the node (if it is not a coordinated instance).

Each passive coordinated instance (those for which a TP_P_ABORT_IND, TP_U_ABORT_IND, TP_U_ABORT_REQ or TP_BEGIN_DIALOGUE_CONF(rejected, Rollback="false") primitive has been issued) returns to the AP_TP_IDLE state on receipt of this primitive.

All dialogues for which a TP_DEFERRED_END_DIALOGUE request or indication was issued during the committed transaction are terminated.

Control of all dialogues with a TP_DEFERRED_GRANT_CONTROL_REQ outstanding has now passed to the receiver of the TP_DEFERRED_GRANT_CONTROL_IND.

The coordination level of all dialogues with the commit and unchained transactions functional unit selected has reverted to none.

If one or more of the dialogues has the commit and chained transactions functional unit selected, then they are involved in a new transaction.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
tp_dialog_env_t *tp_env; /* dialogue attribute values */
```

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-COMMIT-COMPLETE indication service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in *cdata* and *cdata→tp_env* will be set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The dialogue tree node identifier *dtnid* will be set to the value allocated to the TPSUI when it was started. The transaction tree node identifier *ttnid* will only be present if the user has set a value in this attribute for the node.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_DATA_REQ — used to send a U-ASE primitive

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_DATA_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to send user data over an established dialogue.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_DATA_REQ.

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

```
long udata_length;     /* length of              */
                       /* user-information field */
unsigned long user_id; /* U-ASE identifier       */
```

cdata→udata_length may be set to the total number of octets of user-information that will be sent with this primitive. If the amount of data to be sent with this primitive is not known, this field should be set to –1. In some XAP-TP implementations, setting this field may improve performance.

If the U-ASE resides above the XAP-TP interface, *user_id* is not used.

If the U-ASE resides below the XAP-TP interface, *user_id* identifies the U-ASE index number (commencing with 0) within the application context to which the request (or portion of a request) in data is being sent.

ubuf Use of the *ubuf* argument is described on the *ap_snd*( ) manual page. If the U-ASE resides below the XAP-TP interface the data is formatted according to the U-ASE specification. If the U-ASE resides above the XAP-TP interface, the buffer(s) are encoded as:

```
SEQUENCE
{ Transfer-syntax-name OBJECT IDENTIFIER OPTIONAL,
  Presentation-context-identifier INTEGER,
  Presentation-data-values CHOICE
    { single-ASN1-type [0] ANY,
      octet-aligned    [1] IMPLICIT OCTET STRING,
      arbitrary        [2] IMPLICIT BIT STRING}
}
```

Note that a TP_DATA_REQ primitive may not be issued without one or more octets of user data.

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

Note that the TP_DATA_REQ primitive may not be issued without one or more octets of user-data.

RETURN VALUE

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

ERRORS

In addition to the errors listed on the *ap_snd*( ) manual page, the following TP_DATA_REQ error may occur:

[AP_NODATA]
An attempt was made to send this primitive with no user-data.

NAME

TP_DATA_IND — used to indicate receipt of a U-ASE primitive

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_DATA_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the receipt of a U-ASE primitive.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
unsigned long user_id;  /* U-ASE identifier */
```

If the U-ASE resides above the XAP-TP interface, *user_id* is not used.

If the U-ASE resides below the XAP-TP interface, *user_id* identifies the U-ASE index number (commencing with 0) within the application context from which the primitive (or portion of a primitive) in data has been received.

ubuf Use of the *ubuf* argument is described on the *ap_rcv*( ) manual page. If the U-ASE resides below the XAP-TP interface, the data is formatted according to the U-ASE specification. If the U-ASE resides above the XAP-TP interface the buffer(s) are encoded as:

```
SEQUENCE
{ Transfer-syntax-name OBJECT IDENTIFIER OPTIONAL,
  Presentation-context-identifier INTEGER,
  Presentation-data-values CHOICE
    { single-ASN1-type [0] ANY,
      octet-aligned    [1] IMPLICIT OCTET STRING,
      arbitrary        [2] IMPLICIT BIT STRING}
}
```

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

RETURN VALUE

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

ERRORS

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

NAME

TP_DEFERRED_END_DIALOGUE_REQ — request the dialogue end when the transaction is committed

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_DEFERRED_END_DIALOGUE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request that the dialogue is ended when the transaction is committed.

The dialogue must have a coordination level of commitment.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_DEFERRED_END_DIALOGUE_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_DEFERRED_END_DIALOGUE_IND — the dialogue is to be ended when the transaction commits

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_DEFERRED_END_DIALOGUE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the dialogue is to be ended when the transaction is committed.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_DEFERRED_GRANT_CONTROL_REQ — request control pass to other end of dialogue on transaction commitment

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_DEFERRED_GRANT_CONTROL_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request that control pass to the other end of the dialogue on transaction commitment.

The dialogue must have a coordination level of commitment.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_DEFERRED_GRANT_CONTROL_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_DEFERRED_GRANT_CONTROL_IND — indicates that control is to pass on transaction completion

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_DEFERRED_GRANT_CONTROL_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that control is to pass to this end of the dialogue on transaction completion.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_DIALOGUE_LOST_IND — indicates that one of the coordinated dialogues of the node has been lost

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_DIALOGUE_LOST_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that one of the coordinated dialogues of the node has been abnormally terminated.

When a coordinated dialogue is abnormally terminated, all other XAP-TP instances supporting an active coordinated dialogue for the node and the nominated control instance (if this is not also supporting an active coordinated dialogue) receive a TP_DIALOGUE_LOST_IND primitive to allow the user to coordinate failure related actions.

If the transaction is being rolled back as a result of this failure, any in progress send or receive will have been terminated.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long tp_options;          /* rollback / failure details  */
long tp_fail_count;       /* count of failure conditions */
tp_dialog_env_t *tp_env;  /* dialogue attribute values   */
```

cdata→tp_options indicates if the transaction in which the recipient is involved has commenced rollback as a result of the failure, and whether the dialogue which failed was from the superior or to a subordinate. The bit values in *cdata→tp_options* used are:

AP_TP_ROLLBACK
: If set, the transaction has commenced rollback as a result of this dialogue loss. If unset, no rollback is occurring as a result of this dialogue loss or rollback was already in progress at the time of the dialogue loss.

AP_TP_SUPERIOR
: If set, the dialogue lost was from the superior. If unset, it was to a subordinate.

cdata→tp_fail_count holds the number of failure conditions which have occurred on the transaction node since the start of the transaction, and must be used when issuing a TP_DONE_REQ primitive to acknowledge completion of any failure related actions.

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-DIALOGUE-LOST indication service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values are not returned in *cdata* and *cdata→tp_env* is set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* points to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The dialogue tree node identifier *dtnid* will be set to the value allocated to the TPSUI when it was started. The transaction tree node identifier *ttnid* will only be present if the user has set a value in this attribute for the node.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_DONE_REQ — used to indicate that local commitment is complete

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_DONE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to indicate that commitment of local resources, and/or failure-related actions have been completed for the node identified by *dtnid* or *ttnid*.

If not all failure conditions have been taken into account by the user, the TP_DONE_REQ primitive will be refused with the error code [AP_TP_BADCD_FAIL_COUNT]. The user should receive the pending failure indication(s) and resubmit the TP_DATA_REQ primitive with an updated *cdata→tp_fail_count* when any failure-related actions necessary have been completed.

The primitive must be issued on the nominated control instance for the transaction node.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_DONE_REQ.

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

```
long res;                   /* local outcome                */
long tp_fail_count;         /* count of failure conditions  */
tp_dialog_env_t *tp_env;    /* dialogue attribute values    */
```

cdata→res is used to return the results of local commitment or rollback by the TPSUI. The following values may be used:

AP_TP_NONE
: Commit or rollback has completed successfully.

AP_TP_HEUR_MIX
: The bound data handled by the TPSUI are in a state inconsistent with the outcome of the transaction and the inconsistency cannot be corrected.

AP_TP_HEUR_HAZ
: A failure occurred within the TPSUI which may prevent reporting of data inconsistency, and the TPSUI may not handle this situation.

cdata→tp_fail_count holds the number of failure conditions that have occurred on the node for which failure related actions have been completed.

cdata→tp_env can be used to override XAP environment attributes values used as parameters to the TP-DONE request service. If no attribute values are to be

overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The user may identify the transaction using one of:

- the Dialogue Tree Node Identifier AP_DTNID
- the Transaction Tree Node Identifier AP_TTNID.

If the XAP instance has the TP_DIALOGUE category selected, it is only possible to indicate completion of commitment of the current transaction on the dialogue tree referenced by a dialogue on the XAP instance.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

[AP_TP_BADCD_FAIL_COUNT]
The given *tp_fail_count* does not match the XAP-TP provider's count of failure conditions.

[AP_TP_BAD_NODE]
An extant transaction node is not identified by AP_DTNID or AP_TTNID attributes.

NAME

TP_END_DIALOGUE_REQ — used to request ending of the dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_END_DIALOGUE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request ending of the dialogue.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_END_DIALOGUE_REQ.

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

```
long tp_options; /* confirmation requested */
```

cdata→tp_options is used to specify if confirmation of this primitive is required. The possible bit settings of *cdata→tp_options* for this primitive are:

AP_TP_CONFIRM
: If set, the recipient must confirm ending of the dialogue by issuing a TP_END_DIALOGUE_RSP.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

In addition to the errors listed on the *ap_snd*( ) manual page, the following TP_END_DIALOGUE_REQ errors may occur:

[AP_TP_BADCD_TP_OPTIONS]
: The setting of *cdata→tp_options* is invalid.

NAME

TP_END_DIALOGUE_IND — used to indicate ending of a dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_END_DIALOGUE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the dialogue is to be ended.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long tp_options;        /* confirmation requested */
```

cdata→tp_options is used to specify if confirmation of this primitive is required. The following bit values are used for this field:

AP_TP_CONFIRM
: If set, the user must confirm ending of the dialogue by issuing a TP_END_DIALOGUE_RSP.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_END_DIALOGUE_RSP — used to respond to a end dialogue request

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_END_DIALOGUE_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to a TP_END_DIALOGUE_IND which has specifically requested confirmation via the AP_TP_CONFIRM bit setting of *cdatatp_options*.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_END_DIALOGUE_RSP.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_END_DIALOGUE_CNF — used to confirm ending of a dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_END_DIALOGUE_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm that a dialogue has been ended.

Confirmation must have been requested in the TP_END_DIALOGUE_REQ via the AP_TP_CONFIRM bit setting of *cdata→tp_options* in order to receive this primitive.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_FLUSH_REQ — used to flush internally queued pdus

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_FLUSH_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to flush any PDUs which are queued internally due to application of the TP concatenation rules. The PDUs are passed immediately to the presentation layer. The primitive has no effect if no PDUs are queued.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_FLUSH_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_GRANT_CONTROL_REQ — used to pass control of the dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_GRANT_CONTROL_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to pass control of the dialogue away from the sending node.

The dialogue must have the polarised control functional unit selected, and control must currently reside with the sending node.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_GRANT_CONTROL_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_GRANT_CONTROL_IND — used to indicate dialogue control has passed to the user

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_GRANT_CONTROL_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that control of the dialogue has passed to the receiving node.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_HANDSHAKE_REQ — used to request a synchronising handshake

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HANDSHAKE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request a synchronising handshake on the dialogue.

The dialogue must have the handshake functional unit selected, and either the polarised control or shared control functional unit selected.

If the shared control functional unit is selected on the dialogue, the *tp_options* parameter indicates the urgency with which a response is required.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_HANDSHAKE_REQ.

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

```
long tp_options;        /* confirmation requested */
```

cdata→tp_options is used to specify the urgency with which confirmation is required. This argument is only valid if the shared control functional unit is selected on the dialogue. The following bit settings are used in this field:

AP_TP_URGENT
: If set, minimal delay is requested for receiving the TP_HANDSHAKE_CNF to complete the handshake. If unset, there is no particular delay requirement.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

In addition to the errors listed on the *ap_snd*( ) manual page, the following TP_HANDSHAKE_REQ errors may occur:

[AP_TP_BADCD_TP_OPTIONS]
: The setting of *cdata→tp_options* is invalid.

NAME

TP_HANDSHAKE_IND — used to indicate a request for a synchronising handshake

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HANDSHAKE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a request to synchronise processing by a handshake exchange.

If the shared control functional unit is selected on the dialogue and the requestor indicated that the response was urgent, by setting the AP_TP_URGENT bit of the *cdata→tp_options* field, the TPSP will have recorded this, and the handshake response will be sent immediately.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_HANDSHAKE_RSP — used to respond to a handshake request

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HANDSHAKE_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to respond to a handshake request on a dialogue.

If the shared control functional unit is selected on the dialogue and the requestor indicated that the response was urgent, by setting the AP_TP_URGENT bit of the *cdata→tp_options* field when sending the TP_HANDSHAKE_REQ, the TPSP will have recorded this, and the handshake response will sent immediately.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_HANDSHAKE_RSP.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_HANDSHAKE_CNF — confirms a handshake request

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HANDSHAKE_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm completion of a handshake request.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_HANDSHAKE_AND_GRANT_CONTROL_REQ — used to request a synchronising handshake and to grant control

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HANDSHAKE_AND_GRANT_CONTROL_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request synchronising processing by a handshake exchange and to simultaneously grant control of the dialogue.

The dialogue must have the polarised control and handshake functional units selected, and control must currently reside with the sending node.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_HANDSHAKE_AND_GRANT_CONTROL_REQ.

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

```
long tp_options;        /* confirmation requested */
```

cdata→tp_options is used to specify the urgency with which confirmation is required. This argument is only valid if the shared control functional unit is selected on the dialogue. The following bit settings are used in this field:

AP_TP_URGENT
: If set, minimal delay is requested for receiving the TP_HANDSHAKE_AND_GRANT_CONTROL_CNF to complete the handshake. If unset, there is no particular delay requirement.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

In addition to the errors listed on the *ap_snd*( ) manual page, the following TP_HANDSHAKE_AND_GRANT_CONTROL_REQ errors may occur:

[AP_TP_BADCD_TP_OPTIONS]
The setting of *cdata→tp_options* is invalid.

NAME

TP_HANDSHAKE_AND_GRANT_CONTROL_IND — used to request a synchronising handshake and to grant control

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HANDSHAKE_AND_GRANT_CONTROL_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a request to synchronise processing by a handshake exchange and to indicate that control of the dialogue has passed to the receiving node.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_HANDSHAKE_AND_GRANT_CONTROL_RSP — used to respond to a handshake and grant control request

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HANDSHAKE_AND_GRANT_CONTROL_RSP primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to indicate completion of the synchronisation activity requested in a previous handshake and grant control indication on the dialogue.

If the requestor indicated that the response was urgent, by setting the AP_TP_URGENT bit of the *cdata→tp_options* field when sending the TP_HANDSHAKE_AND_GRANT_CONTROL_REQ, the TPSP will have recorded this, and the response will sent immediately.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_HANDSHAKE_AND_GRANT_CONTROL_RSP.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_HANDSHAKE_AND_GRANT_CONTROL_CNF — used to confirm completion of a synchronising handshake and grant control

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HANDSHAKE_AND_GRANT_CONTROL_CNF primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to confirm completion of a synchronising handshake and grant control request.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_HEURISTIC_REPORT_IND — indicates a possible or actual heuristic condition

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_HEURISTIC_REPORT_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a possible or actual heuristic inconsistency within the subordinate subtree below the dialogue.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long res;              /* type of inconsistency */
```

cdata→res is used to indicate the heuristic condition. The following values are legal for this field:

AP_TP_HEUR_MIX
: The state of the data of the subordinate subtree is inconsistent with the outcome of the transaction, and the inconsistency cannot be corrected.

AP_TP_HEUR_HAZ
: A failure has occurred which may prevent reporting of data inconsistency in the subordinate subtree.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_LOG_DAMAGE_IND — used to log a heuristic condition in the subtree

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_LOG_DAMAGE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to update a ready or commit log record with information about possible or actual heuristic inconsistency in the data of the subtree. The log record to be updated is specified by *dtnid* and *ttnid* (if set).

This primitive will only be issued on the nominated control instance for the transaction node.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-LOG-DAMAGE indication service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in *cdata* and *cdata→tp_env* will be set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The dialogue tree node identifier *dtnid* will be set to the value allocated to the TPSUI when it was started. The transaction tree node identifier *ttnid* will only be present if the user has set a value in this attribute for the node.

ubuf Updated log record.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_MANAGE_REQ — changes the nominated control instance of a transaction node

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_MANAGE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to change the nominated control instance of a transaction node. It is issued on the control instance to which the transaction node is to be transferred.

If successful, the XAP-TP provider queues a TP_NODE_STATUS_IND primitive on the instance. This TP_NODE_STATUS_IND primitive returns the current state of the node and current log record (if any) to enable the user to synchronise processing with the XAP-TP provider.

Note that due to the asynchronous nature of the XAP-TP interface, previously queued commit and log primitives may still be received on the original control instance (if still available). These primitives may be safely ignored.

If the transaction node, whose nominated control instance is to be changed, has already completed commitment and has queued a TP_COMMIT_COMPLETE_IND for the user, it may have ceased to exist. If it no longer exists the user will receive the [AP_TP_BAD_NODE] error code, from which they can infer that the transaction node has successfully completed commitment and so they may safely delete any associated log records.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_MANAGE_REQ.

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to override XAP environment attributes values used as parameters to the TP-MANAGE request service. If no attribute values are to be overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

The *mask* element of this structure is a bit mask indicating which environment attributes associated with this primitive are to be overridden. Values for this field

are formed by setting zero or one of the flags listed in the table below. When a bit is set, the value of the associated parameter will be taken from *cdata* rather than from the XAP environment. Specifying a value for a particular parameter in *cdata* has the same effect on the value of the corresponding attribute in the XAP environment as calling *ap_set_env*( ).

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The user may identify the transaction node whose control instance is being changed by either:

- the Dialogue Tree Node Identifier AP_DTNID
- the Transaction Tree Node Identifier AP_TTNID.

If the XAP instance has the TP_DIALOGUE category selected, the instance can only become the control instance for a transaction node current on the dialogue tree node referenced by the XAP instance.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

In addition to the errors listed on the *ap_snd*( ) manual page, the following errors may occur:

[AP_TP_BAD_URCH]
The node does not belong to the same recovery context group as the XAP-TP instance.

[AP_TP_BAD_NODE]
AP_TTNID or AP_DTNID does not identify an existing node.

NAME

TP_NODE_STATUS_IND — returns the status of a transaction node

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_NODE_STATUS_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to return details of the current state and log record (if any) of a transaction node. It is received during control instance resumption or as a result of a TP_MANAGE_REQ to change the control instance of a node.

This primitive will only be issued on the nominated control instance for the transaction node.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long tp_fail_count;        /* count of failure conditions  */
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_fail_count holds the number of failure conditions which have occurred on the transaction node since the start of the transaction, and must be used when issuing a TP_DONE_REQ primitive to acknowledge completion of any failure related actions. The user can determine whether a TP_DONE_REQ primitive is required for the node, to acknowledge completion of failure related actions, by examining the node's state returned in AP_TP_STATE attribute.

cdata→tp_env can be used to retrieve the values of the XAP environment attributes for the node whose details are being returned. If the AP_TP_COPYENV attribute in the XAP environment is *false*, these values will not be returned in *cdata* and *cdata→tp_env* will be set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is *true*, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask        */
ap_aaid_t aaid;              /* AP_AAID         */
ap_brid_t brid;              /* AP_BRID         */
ap_dtnid_t dtnid;            /* AP_DTNID        */
ap_ttnid_t ttnid;            /* AP_TTNID        */
unsigned long tp_state;      /* AP_TP_STATE     */
```

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

Flag	Parameter	Field
AP_AAID_BIT	Atomic Action Identifier	*aaid*
AP_BRID_BIT	Branch Identifier	*brid*
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*
AP_TP_STATE_BIT	Transaction Tree Node State	*tp_state*

The dialogue tree node identifier *dtnid* will be set to the value allocated to the TPSUI when it was started. The transaction tree node identifier *ttnid* will only be present if the user has set a value in this attribute for the node.

ubuf Last issued log record for the node. NULL if none has been issued.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_P_ABORT_IND — used to indicate the failure of a dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_P_ABORT_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the TP service provider has detected abnormal termination of the dialogue.

If the dialogue was coordinated, the nodes updated failure count is returned and the failure is propagated to the nominated control instance for the node, and to each other active coordinated instance of the node, via a TP_DIALOGUE_LOST_IND primitive.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long diag;           /* reason for failure          */
long tp_options;     /* rollback transaction?       */
long tp_fail_count;  /* count of failure conditions */
```

cdata→diag indicates the reason for the failure. The possible values are:

AP_TP_PERMANENT
: A permanent error condition has been encountered.

AP_TP_TRANSIENT
: A transient error condition has been encountered.

AP_TP_PROTOCOL_ERROR
: A protocol error has been encountered.

AP_TP_REJ_TRANSACTION
: A TP_BEGIN_TRANSACTION_IND primitive was not issued because the recipient is already involved in a transaction or because of a local condition.

AP_TP_END_CLASH
: Two TP_END_DIALOGUE primitives with the confirmation parameter have collided.

AP_TP_BEG_END_CLASH
A TP_BEGIN_TRANSACTION_REQ and a TP_END_DIALOGUE_REQ primitive have collided.

cdata→tp_options indicates if the transaction in which the recipient is involved is to be rolled back. The bit value in *cdata→tp_options* used is:

AP_TP_ROLLBACK
If set, the transaction is to be rolled back. If unset, no rollback is to occur or rollback is already in progress.

cdata→tp_fail_count holds the number of failure conditions which have occurred on the transaction node since the start of the transaction, and must be used when issuing a TP_DONE_REQ primitive to acknowledge completion of any failure related actions. Only valid if the dialogue was coordinated.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_PREPARE_REQ — used to request preparation of a transaction branch

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_PREPARE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request a subordinate subtree to complete processing for the current transaction and place its data in the ready to commit state.

The dialogue must be to a subordinate.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_PREPARE_REQ.

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

```
long tp_options;        /* data permitted ?    */
```

cdata→tp_options is used to specify if the receiving node can issue further data requests. The following bit settings are used in this field:

AP_TP_PERMITTED
The receiving node can issue further data requests.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

In addition to the errors listed on the *ap_snd*( ) manual page, the following TP_PREPARE_REQ errors may occur:

[AP_TP_BADCD_TP_OPTIONS]
The setting of *cdata→tp_options* is invalid.

NAME

TP_PREPARE_IND — used to indicate a request to commit a transaction

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_PREPARE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a request to the application to prepare for commitment of the current transaction.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long tp_options;        /* data permitted ?    */
```

cdata→tp_options is used to specify if the receiving node can issue further data requests. The following bit settings are used in this field:

AP_TP_PERMITTED
: The receiving node can issue further data requests.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_PREPARE_ALL_REQ — used to perform the first phase of commit

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_PREPARE_ALL_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to ensure that the entire subordinate subtree of the node is brought to the ready to commit state.

It causes a prepare request to be issued on each subordinate dialogue that has not had an explicit TP_PREPARE_REQ issued.

When the first phase is complete, a TP_READY_ALL_IND is issued indicating the nodes subtree is in the ready to commit state.

Note that this primitive must be issued even at a leaf node (which has no subordinate dialogues) so that a ready record is logged by the transaction manager.

The primitive must be issued on the nominated control instance for the transaction node.

The *dtnid* or *ttnid* arguments identify the node whose subtree is to be prepared.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_PREPARE_ALL_REQ.

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to override XAP environment attributes values used as parameters to the TP-PREPARE-ALL request service. If no attribute values are to be overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

The *mask* element of this structure is a bit mask indicating which environment attributes associated with this primitive are to be overridden. Values for this field are formed by setting zero or one of the flags listed in the table below. When a bit is set, the value of the associated parameter will be taken from *cdata* rather than

from the XAP environment. Specifying a value for a particular parameter in *cdata* has the same effect on the value of the corresponding attribute in the XAP environment as calling *ap_set_env*( ).

flag	parameter	field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The user may identify the transaction to be prepared using one of:

- the Dialogue Tree Node Identifier AP_DTNID
- the Transaction Tree Node Identifier AP_TTNID.

If the XAP instance has the TP_DIALOGUE category selected and is within a dialogue, it is only possible to request preparation of the subtree belonging to the node to which the dialogue is attached.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

[AP_TP_BAD_NODE]
An extant transaction node is not identified by AP_DTNID or AP_TTNID.

NAME

TP_READY_IND — used to indicate a subtree is ready to commit

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_READY_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the subordinate subtree has reached the ready to commit state.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_READY_ALL_IND — used to indicate all subordinates are ready to commit

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_READY_ALL_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that all subordinate dialogs have reported ready to commit.

The *dtnid* or *ttnid* indicate which node is reporting ready to commit.

This primitive will only be issued on the nominated control instance for the transaction node.

At an intermediate or leaf node the transaction manager must secure the log record prior to reporting its willingness to participate in commitment by issuing a TP_COMMIT_REQ. At a root node, the transaction manager must secure the log record prior to instructing the node to commit by issuing a TP_COMMIT_REQ primitive (typically it would convert the log record into a log-commit record prior to securing it). If the transaction manager chooses instead to rollback the node, it issues a TP_ROLLBACK_REQ primitive without having secured the log record.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-READY-ALL indication service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in *cdata* and *cdata→tp_env* will be set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

the corresponding field in the **tp_dialog_env_t** structure is not set.

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The dialogue tree node identifier *dtnid* will be set to the value allocated to the TPSUI when it was started. The transaction tree node identifier *ttnid* will only be present if the user has set a value in this attribute for the node.

ubuf Log ready record.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_RECOVER_REQ — used to pass current log records to XAP-TP

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_RECOVER_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to pass a current log record to XAP-TP during a recovery context group restart.

This primitive can only be issued on one of the control instances within the recovery context group. If processing of a log record would result in two transaction nodes with the same TTNID, the log record is refused with the error code [AP_TP_BAD_TTNID].

If a TPPM is successfully reconstructed, the DTNID of the node is returned.

If the user passes a log record which does not belong to the bound AE-title or to the bound recovery context group, the XAP-TP provider will detect it and refuse the log record with the error code [AP_TP_BAD_LOG]. No TPPM will be constructed for the log record, and the user may continue to submit further log records in later calls.

For a log-ready or log-commit record, XAP-TP reconstructs a TPPM. For a log-damage record, XAP-TP establishes an internal heuristic_damage entry. Recovery for a reconstructed TPPM will not commence until the restart is successfully completed.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_RECOVER_REQ.

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

```
tp_dialog_env_t *tp_env; /* dialogue attribute values    */
```

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of the TP-RECOVER request service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in *cdata*, and *cdata→tp_env* is set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* points to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;         /* bit mask */
ap_dtnid_t dtnid;           /* AP_DTNID */
ap_ttnid_t ttnid;           /* AP_TTNID */
```

The mask element of this structure is a bit mask indicating which parameters associated with this primitive were returned. Values for this field are formed by

setting zero or one of the flags listed in the table below. When a bit is set, the specified parameter is present. Otherwise, the parameter was not received and the corresponding field in the **tp_dialog_env_t** structure is not set.

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*dtnid*

The dialogue tree node identifier is set to the newly allocated value. The transaction tree node identifier will only be present if it was present in the log record.

ubuf The log record. Note that the log record will have been passed to the transaction manager by a previous TP_READY_ALL_IND or TP_LOG_DAMAGE_IND primitive.

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

RETURN VALUE

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

ERRORS

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

[AP_TP_BAD_LOG]
The log record supplied does not belong to AE-title the instance is bound to, or does not belong to the recovery context group the instance is bound to.

[AP_TP_BAD_TTNID]
The TTNID in the log record supplied is already in use by another transaction node in the recovery context group.

NAME

TP_REQUEST_CONTROL_REQ — used to request control of the dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_REQUEST_CONTROL_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to request control of the dialogue be passed to the caller.

The primitive can only be issued if the polarised control functional unit is selected and control currently resides with the remote end of the dialogue.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_REQUEST_CONTROL_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_REQUEST_CONTROL_IND — indicates a request for control of the dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_REQUEST_CONTROL_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate a request for control of the dialogue.

The caller should pass control of the dialogue at a suitable point by calling either TP_GRANT_CONTROL_REQ or TP_HANDSHAKE_AND_GRANT_CONTROL_REQ.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_RESUME_REQ — begins resuming use of a control instance

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_RESUME_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to begin resuming use of a control instance within a recovery context group.

This must be the first primitive on a control instance after it is established.

If the recovery context group is currently unavailable, the TP_RESUME_REQ will be rejected with the error code [AP_TP_RESTART_REQD]. The user must then perform a restart of the recovery context group before the control instance can be used.

If the recovery context group is currently being restarted, the primitive will be rejected with the error code [AP_TP_RESTARTING]. A resume must be performed for a recovery context group before it can be used to process incoming and outgoing begin dialogues with the commit functional unit selected. XAP-TP will respond retry-later to all incoming recovery requests for the recovery context group until the resume has been successfully completed.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_RESUME_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

[AP_TP_RESTART_REQD]
The recovery context group is currently unavailable.

[AP_TP_RESTARTING]
The recovery context group is currently restarting.

NAME

TP_RESUME_COMPLETE_IND — indicates control instance resumption is complete

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_RESUME_COMPLETE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate resumption of control instance usage within a recovery context group has been successfully completed. Normal usage of the control instance commences after receipt of this primitive.

This primitive is issued on the control instance whose resume has ended.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_RESTART_REQ — begins the restart of a recovery context group

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_RESTART_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to begin the restart of a recovery context group.

This primitive can only be issued on a control instance within the recovery context group.

A restart must be performed for a recovery context group before it can be used to process incoming and outgoing begin dialogues with the commit functional unit selected. XAP-TP will respond retry-later to all incoming recovery requests for the recovery context group until the restart has been successfully completed.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_RESTART_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_RESTART_COMPLETE_REQ — ends a recovery context group restart

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_RESTART_COMPLETE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to indicate to XAP TP that the restart of the recovery context group is complete on this XAP-TP instance. Once this primitive has been issued on all control instances actively participating in restart, XAP TP will issue a TP_RESTART_COMPLETE_IND primitive to each control instance of the recovery context group and commence normal operation responding to incoming recovery and dialogue requests.

This primitive can only be issued on a control instance within the recovery context group.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_RESTART_COMPLETE_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_RESTART_COMPLETE_IND — indicates end of restart for a control instance

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_RESTART_COMPLETE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that restart of a recovery context group has been successfully completed. Normal usage of the control instance commences after receipt of this primitive.

This primitive is issued on each control instance of the recovery context group that has been actively or passively participating in the restart.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long res;     /* result of restart */
```

cdata→res indicates the outcome of the restart, and can take the following values:

AP_TP_ACCEPT
: The restart completed successfully.

AP_TP_REJ_PROV
: The restart was aborted by the XAP-TP provider.

AP_TP_REJ_USER
: The restart was aborted by the user.

If *cdata→res* is AP_TP_ACCEPT, the state will have changed to AP_TP_IDLE after receiving this primitive, otherwise the state will have changed to AP_TP_WRESUMEreq.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_ROLLBACK_REQ — used to request roll back of a transaction

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_ROLLBACK_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to terminate the transaction tree node identified by *dtnid* or *ttnid*, and set the data into the initial state.

The primitive must be issued on the nominated control instance for the transaction node.

The rollback is propagated to each active coordinated instance of the node (except when it is also the nominated control instance). Each receives a TP_ROLLBACK_IND primitive to indicate that the transaction node is rolling back. This allows the user to coordinate failure-related actions between the instances.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_ROLLBACK_REQ.

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to override XAP environment attributes values used as parameters to the TP-ROLLBACK request service. If no attribute values are to be overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The user may identify the transaction to be rolled back using one of:

- the Dialogue Tree Node Identifier AP_DTNID
- the Transaction Tree Node Identifier AP_TTNID.

If the XAP instance has the TP_DIALOGUE category selected and is within a dialogue, it is only possible to request rollback of a transaction tree node on the TPSUI to which this dialogue is attached.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

[AP_TP_BAD_NODE]
An extant transaction node is not identified by the AP_DTNID or AP_TTNID attributes.

NAME

TP_ROLLBACK_IND — indicates a transaction is being rolled back

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_ROLLBACK_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the transaction tree node identified by *dtnid* or *ttnid* is being rolled back.

This primitive is issued on the nominated control instance for the transaction node, and on each active coordinated instance of the node (except when it is also the nominated control instance). This allows the user to coordinate failure related actions between the instances.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long tp_fail_count;        /* count of failure conditions  */
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_fail_count holds the number of failure conditions which have occurred on the transaction node since the start of the transaction (in this case one), and must be used when issuing a TP_DONE_REQ primitive to acknowledge completion of any failure-related actions.

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-ROLLBACK indication service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in *cdata* and *cdata→tp_env* will be set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

the corresponding field in the **tp_dialog_env_t** structure is not set.

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The dialogue tree node identifier *dtnid* will be set to the value allocated to the TPSUI when it was started. The transaction tree node identifier *ttnid* will only be present if the user has set a value in this attribute for the node.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_ROLLBACK_COMPLETE_IND — indicates completion of a transaction rollback

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_ROLLBACK_COMPLETE_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the transaction tree node identified by *dtnid* or *ttnid* has completed being rolled back.

This primitive is passed to the user on each coordinated instance of the node, and also on the nominated control instance for the node (if it is not a coordinated instance).

Each passive coordinated instance (those for which a TP_P_ABORT_IND, TP_U_ABORT_IND, TP_U_ABORT_REQ or TP_BEGIN_DIALOGUE_CONF(rejected) primitive has been issued) returns to the AP_TP_IDLE state on receipt of this primitive.

The coordination level of all dialogues with the commit and unchained functional unit selected reverts to none.

If one or more of the dialogues has the commit and chained functional unit selected, then they are involved in another transaction.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to retrieve the values of the XAP environment attributes that correspond to parameters of TP-ROLLBACK-COMPLETE indication service. If the AP_TP_COPYENV attribute in the XAP environment is FALSE, these values will not be returned in *cdata* and *cdata→tp_env* will be set to NULL when *ap_rcv*( ) returns. If AP_TP_COPYENV is TRUE, *cdata→tp_env* will point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_dtnid_t dtnid;            /* AP_DTNID   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

The *mask* element of this structure is a bit mask indicating which parameters associated with this primitive were received. Values for this field are formed by

setting zero or one of the flags listed in the table below. When a bit is set, the specified parameter was received. Otherwise, the parameter was not received and the corresponding field in the **tp_dialog_env_t** structure is not set.

Flag	Parameter	Field
AP_DTNID_BIT	Dialogue Tree Node Identifier	*dtnid*
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The dialogue tree node identifier *dtnid* will be set to the value allocated to the TPSUI when it was started. The transaction tree identifier *ttnid* will only be present if the user has set a value in this attribute for the node.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_UPDATE_LOG_DAMAGE_REQ — used to update or forget locally-held heuristic damage information

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_UPDATE_LOG_DAMAGE_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to update or remove local knowledge of a heuristic log-damage record for the transaction node specified by *ttnid*.

XAP-TP maintains an internal copy of heuristic log-damage information until removed by TP_UPDATE_LOG_DAMAGE_REQ. This is used to regenerate heuristic reports in response to recovery request from superiors.

This primitive can only be issued on the nominated control instance for the transaction node.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_UPDATE_LOG_DAMAGE_REQ

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

```
tp_dialog_env_t *tp_env;   /* dialogue attribute values    */
```

cdata→tp_env can be used to override XAP environment attributes values used as parameters to the TP-UPDATE-LOG-DAMAGE request service. If no attribute values are to be overridden, *cdata→tp_env* may be set to NULL. Otherwise, *cdata→tp_env* must point to a **tp_dialog_env_t** structure, and the following elements are used for this primitive:

```
unsigned long mask;          /* bit mask   */
ap_ttnid_t ttnid;            /* AP_TTNID   */
```

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

Flag	Parameter	Field
AP_TTNID_BIT	Transaction Tree Node Identifier	*ttnid*

The user can identify the transaction using the Transaction Tree Node Identifier AP_TTNID.

Note that it is only possible to use this primitive on an XAP instance which has the TP_DIALOGUE category selected if it is not currently in use for a dialogue.

ubuf If set to **NULL**, the log-damage details for the specified transaction node are deleted. Otherwise the contents of *ubuf* buffers must be an updated log-damage record.

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

RETURN VALUE

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

ERRORS

In addition to the errors listed in the manual page for *ap_snd*( ), the following error conditions can be reported for this primitive:

[AP_TP_BAD_NODE]
An extant transaction node is not identified by the AP_DTNID or AP_TTNID attributes.

NAME

TP_U_ABORT_REQ — used to terminate a dialogue with user data

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_U_ABORT_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to abort the dialogue passing application supplied data.

If the dialogue is coordinated and a failure condition is not outstanding, the resulting dialogue loss will be propagated to the control instance for the node (if this is not the originating instance) and to all other active coordinated instances of the node via TP_DIALOGUE_LOST_IND primitives.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_U_ABORT_REQ.

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

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

cdata→udata_length may be set to the total number of octets of user-information that will be sent with this primitive. If the amount of data to be sent with this primitive is not known, this field should be set to –1. In some XAP-TP implementations, setting this field may improve performance.

ubuf Use of the *ubuf* argument is described on the *ap_snd*( ) manual page. Data carried in the *ubuf* buffer(s) must be encoded according to the definition specified in ISO / IEC 10026-3: 1992 (the OSI TP Protocol) ([30] IMPLICIT SEQUENCE OF EXTERNAL).

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

RETURN VALUE

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

ERRORS

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

NAME

TP_U_ABORT_IND — indicates a remote user abort of the dialogue

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_U_ABORT_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate that the remote user has aborted the dialogue.

If the dialogue was coordinated, the nodes updated failure count will be returned and the failure will be propagated to the nominated control instance for the node, and to each active coordinated instance of the node (except when it is also the nominated control instance), via TP_DIALOGUE_LOST_IND primitives.

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

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

fd This argument identifies the XAP Library instance being used.

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

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

```
long tp_options;       /* rollback transaction?       */
long tp_fail_count;    /* count of failure conditions */
```

The argument *cdata→tp_options* indicates if the transaction in which the recipient is involved is to be rolled back. The bit value in *cdata→tp_options* used is:

AP_TP_ROLLBACK
: If set, the transaction is to be rolled back. If unset, no rollback is to occur or rollback is already in progress

cdata→tp_fail_count holds the number of failure conditions which have occurred on the transaction node since the start of the transaction, and must be used when issuing a TP_DONE_REQ primitive to acknowledge completion of any failure related actions. Only valid if the dialogue was coordinated.

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

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

RETURN VALUE

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

ERRORS

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

NAME

TP_U_ERROR_REQ — used to report a user detected processing error

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_U_ERROR_REQ primitive is used in conjunction with *ap_snd*( ) and the XAP Library environment to report a user-detected processing error. Also serves as a negative response to a TP_HANDSHAKE_IND, a TP_HANDSHAKE_AND_GRANT_CONTROL_IND, and to a TP_END_DIALOGUE_IND requiring confirmation.

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

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

fd This argument identifies the XAP Library instance being used.

sptype This argument must be set to TP_U_ERROR_REQ.

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

NAME

TP_U_ERROR_IND — indicates a user reported processing error

SYNOPSIS

```
#include <xap_tp.h>

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

DESCRIPTION

The TP_U_ERROR_IND primitive is used in conjunction with *ap_rcv*( ) and the XAP Library environment to indicate the remote user reporting a processing error. Also serves as a negative response to a TP_HANDSHAKE_REQ, a TP_HANDSHAKE_AND_GRANT_CONTROL_REQ, and to a TP_END_DIALOGUE_REQ requiring confirmation.

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

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

fd This argument identifies the XAP Library instance being used.

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

cdata Not used.

ubuf Not used.

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

RETURN VALUE

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

ERRORS

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

Appendix A

XAP-TP Header File

```
/*
 * xap_tp.h
 */

#define XAP_TP
#ifndef AP_ID
#include <xap.h>
#endif

/*
 * XAPTP definitions
 */

typedef struct {
    int size;              /* buffer size in bytes                       */
    unsigned char *udata;  /* buffer with user encoded                   */
                           /* ATOMIC-ACTION-IDENTIFIER                   */
} ap_aaid_t;

typedef struct {
    int size;              /* buffer size in bytes                       */
    unsigned char *udata;  /* buffer with user encoded BRANCH-IDENTIFIER */
} ap_brid_t;

typedef struct {
    int size;              /* control identifier length in bytes         */
    unsigned char *udata;  /* buffer with control identifier             */
} ap_cid_t;

typedef struct {
    int size;              /* buffer size in bytes                       */
    unsigned char *udata;  /* buffer with Dialogue Tree Node Identifier  */
} ap_dtnid_t;

typedef struct {
    int size;              /* buffer size in bytes                       */
    unsigned char *udata;  /* buffer with Transaction Tree Node Id.      */
} ap_ttnid_t;

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

typedef struct {
    int size;              /* recovery context handle length in bytes    */
    unsigned char *udata;  /* buffer with user-assigned recovery         */
                           /* context handle                             */
} ap_urch_t;
```

```
/* TP Address */

typedef struct {
    ap_apt_t apt;          /* application process title         */
    ap_aei_api_id_t apid;  /* application process invocation id */
    ap_aeq_t aeq;          /* application entity qualifier      */
    ap_aei_api_id_t aeid;  /* application entity invocation id  */
    long n_tpsuts;         /* number of TP-User titles          */
    ap_tpsut_t *tpsuts;    /* array of TP-User titles           */
} ap_tpaddr_t;

/* Environment attributes override */

typedef struct {
  unsigned long mask;           /* mask                                    */
  ap_aaid_t aaid;               /* atomic action identifier                */
  ap_brid_t brid;               /* branch identifier                       */
  ap_dtnid_t dtnid;             /* dialogue tree identifier                */
  ap_ttnid_t ttnid;             /* transaction identifier                  */
  ap_apt_t lcl_apt;             /* local application process title         */
  ap_aei_api_id_t lcl_apid;     /* local application process invocation id  */
  ap_aeq_t lcl_aeq;             /* local application entity qualifier      */
  ap_aei_api_id_t lcl_aeid;     /* local application entity invocation id  */
  ap_tpsut_t lcl_tpsut;         /* local TP-User title                     */
  ap_apt_t rem_apt;             /* remote application process title        */
  ap_aei_api_id_t rem_apid;     /* remote application process invocation id */
  ap_aeq_t rem_aeq;             /* remote application entity qualifier     */
  ap_aei_api_id_t rem_aeid;     /* remote application entity invocation id  */
  ap_tpsut_t rem_tpsut;         /* remote TP-User title                    */
  unsigned long tpfu_sel;       /* TP functional units selected           */
  unsigned long tp_version_sel; /* TP version selected                     */
  unsigned long tp_state;       /* transaction node state                  */
} tp_dialog_env_t;

/* OSI-TP Identifier */

#define AP_TP_ID (15)

/* Values for AP_TP_AVAIL */

#define AP_TPVER1 1 /* protocol version */

/* Primitives */

#define AP_TP_WRITESIDE (1 << 8)
#define AP_TP_READSIDE  (3 << 8)
#define AP_TP_MASK      (AP_TP_ID << 16)

#define AP_TP_BEGIN_DIALOGUE_REQ              (AP_TP_MASK|AP_TP_WRITESIDE|0x00)
#define AP_TP_BEGIN_DIALOGUE_RSP              (AP_TP_MASK|AP_TP_WRITESIDE|0x01)
#define AP_TP_BEGIN_TRANSACTION_REQ           (AP_TP_MASK|AP_TP_WRITESIDE|0x02)
#define AP_TP_COMMIT_REQ                      (AP_TP_MASK|AP_TP_WRITESIDE|0x03)
#define AP_TP_DEFERRED_END_DIALOGUE_REQ       (AP_TP_MASK|AP_TP_WRITESIDE|0x05)
#define AP_TP_DEFERRED_GRANT_CONTROL_REQ      (AP_TP_MASK|AP_TP_WRITESIDE|0x06)
#define AP_TP_DONE_REQ                        (AP_TP_MASK|AP_TP_WRITESIDE|0x07)
#define AP_TP_END_DIALOGUE_REQ                (AP_TP_MASK|AP_TP_WRITESIDE|0x08)
#define AP_TP_END_DIALOGUE_RSP                (AP_TP_MASK|AP_TP_WRITESIDE|0x09)
#define AP_TP_FLUSH_REQ                       (AP_TP_MASK|AP_TP_WRITESIDE|0x0a)
#define AP_TP_GRANT_CONTROL_REQ               (AP_TP_MASK|AP_TP_WRITESIDE|0x0b)
```

```
#define AP_TP_HANDSHAKE_REQ                    (AP_TP_MASK|AP_TP_WRITESIDE|0x0c)
#define AP_TP_HANDSHAKE_RSP                    (AP_TP_MASK|AP_TP_WRITESIDE|0x0d)
#define AP_TP_HANDSHAKE_AND_GRANT_CONTROL_REQ  (AP_TP_MASK|AP_TP_WRITESIDE|0x0e)
#define AP_TP_HANDSHAKE_AND_GRANT_CONTROL_RSP  (AP_TP_MASK|AP_TP_WRITESIDE|0x0f)
#define AP_TP_DATA_REQ                         (AP_TP_MASK|AP_TP_WRITESIDE|0x10)
#define AP_TP_MANAGE_REQ                       (AP_TP_MASK|AP_TP_WRITESIDE|0x11)
#define AP_TP_PREPARE_REQ                      (AP_TP_MASK|AP_TP_WRITESIDE|0x15)
#define AP_TP_PREPARE_ALL_REQ                  (AP_TP_MASK|AP_TP_WRITESIDE|0x16)
#define AP_TP_RESUME_REQ                       (AP_TP_MASK|AP_TP_WRITESIDE|0x17)
#define AP_TP_RESTART_REQ                      (AP_TP_MASK|AP_TP_WRITESIDE|0x18)
#define AP_TP_RECOVER_REQ                      (AP_TP_MASK|AP_TP_WRITESIDE|0x19)
#define AP_TP_RESTART_COMPLETE_REQ             (AP_TP_MASK|AP_TP_WRITESIDE|0x1a)
#define AP_TP_UPDATE_LOG_DAMAGE_REQ            (AP_TP_MASK|AP_TP_WRITESIDE|0x1b)
#define AP_TP_REQUEST_CONTROL_REQ              (AP_TP_MASK|AP_TP_WRITESIDE|0x1c)
#define AP_TP_ROLLBACK_REQ                     (AP_TP_MASK|AP_TP_WRITESIDE|0x1d)
#define AP_TP_U_ABORT_REQ                      (AP_TP_MASK|AP_TP_WRITESIDE|0x1e)
#define AP_TP_U_ERROR_REQ                      (AP_TP_MASK|AP_TP_WRITESIDE|0x1f)

#define AP_TP_BEGIN_DIALOGUE_IND               (AP_TP_MASK|AP_TP_READSIDE|0x00)
#define AP_TP_BEGIN_DIALOGUE_CNF               (AP_TP_MASK|AP_TP_READSIDE|0x01)
#define AP_TP_BEGIN_TRANSACTION_IND            (AP_TP_MASK|AP_TP_READSIDE|0x02)
#define AP_TP_COMMIT_IND                       (AP_TP_MASK|AP_TP_READSIDE|0x03)
#define AP_TP_COMMIT_COMPLETE_IND              (AP_TP_MASK|AP_TP_READSIDE|0x04)
#define AP_TP_DEFERRED_END_DIALOGUE_IND        (AP_TP_MASK|AP_TP_READSIDE|0x05)
#define AP_TP_DEFERRED_GRANT_CONTROL_IND       (AP_TP_MASK|AP_TP_READSIDE|0x06)
#define AP_TP_END_DIALOGUE_IND                 (AP_TP_MASK|AP_TP_READSIDE|0x08)
#define AP_TP_END_DIALOGUE_CNF                 (AP_TP_MASK|AP_TP_READSIDE|0x09)
#define AP_TP_GRANT_CONTROL_IND                (AP_TP_MASK|AP_TP_READSIDE|0x0b)
#define AP_TP_HANDSHAKE_IND                    (AP_TP_MASK|AP_TP_READSIDE|0x0c)
#define AP_TP_HANDSHAKE_CNF                    (AP_TP_MASK|AP_TP_READSIDE|0x0d)
#define AP_TP_HANDSHAKE_AND_GRANT_CONTROL_IND  (AP_TP_MASK|AP_TP_READSIDE|0x0e)
#define AP_TP_HANDSHAKE_AND_GRANT_CONTROL_CNF  (AP_TP_MASK|AP_TP_READSIDE|0x0f)
#define AP_TP_DATA_IND                         (AP_TP_MASK|AP_TP_READSIDE|0x10)
#define AP_TP_NODE_STATUS_IND                  (AP_TP_MASK|AP_TP_READSIDE|0x11)
#define AP_TP_HEURISTIC_REPORT_IND             (AP_TP_MASK|AP_TP_READSIDE|0x12)
#define AP_TP_LOG_DAMAGE_IND                   (AP_TP_MASK|AP_TP_READSIDE|0x13)
#define AP_TP_P_ABORT_IND                      (AP_TP_MASK|AP_TP_READSIDE|0x14)
#define AP_TP_PREPARE_IND                      (AP_TP_MASK|AP_TP_READSIDE|0x15)
#define AP_TP_RESTART_COMPLETE_IND             (AP_TP_MASK|AP_TP_READSIDE|0x18)
#define AP_TP_READY_IND                        (AP_TP_MASK|AP_TP_READSIDE|0x1a)
#define AP_TP_READY_ALL_IND                    (AP_TP_MASK|AP_TP_READSIDE|0x1b)
#define AP_TP_REQUEST_CONTROL_IND              (AP_TP_MASK|AP_TP_READSIDE|0x1c)
#define AP_TP_ROLLBACK_IND                     (AP_TP_MASK|AP_TP_READSIDE|0x1d)
#define AP_TP_ROLLBACK_COMPLETE_IND            (AP_TP_MASK|AP_TP_READSIDE|0x1e)
#define AP_TP_U_ABORT_IND                      (AP_TP_MASK|AP_TP_READSIDE|0x20)
#define AP_TP_U_ERROR_IND                      (AP_TP_MASK|AP_TP_READSIDE|0x21)
#define AP_TP_RESUME_COMPLETE_IND              (AP_TP_MASK|AP_TP_READSIDE|0x22)
#define AP_TP_DIALOGUE_LOST_IND                (AP_TP_MASK|AP_TP_READSIDE|0x2a)

/* Define APM messages */

#define AP_APM_ALLOCATE_REQ                   (AP_TP_MASK|AP_TP_WRITESIDE|0x2c)
#define AP_APM_ALLOCATE_CNF                   (AP_TP_MASK|AP_TP_READSIDE  |0x26)
#define AP_APM_ASSOCIATION_LOST_IND           (AP_TP_MASK|AP_TP_READSIDE  |0x27)

/* Attributes */

#define AP_BIND_TPADDR ((AP_TP_ID << 16)|0x00)  /* bind TP address          */
#define AP_AAID        ((AP_TP_ID << 16)|0x01)  /* atomic action identifier */
```

```
#define AP_BRID        ((AP_TP_ID << 16)|0x02)  /* branch identifier        */
#define AP_DTNID       ((AP_TP_ID << 16)|0x03)  /* dialogue tree identifier */
#define AP_TTNID       ((AP_TP_ID << 16)|0x04)  /* transaction identifier   */
#define AP_LCL_APT     ((AP_TP_ID << 16)|0x05)  /* local apt                */
#define AP_LCL_APID    ((AP_TP_ID << 16)|0x06)  /* local apid               */
#define AP_LCL_AEQ     ((AP_TP_ID << 16)|0x07)  /* local aeq                */
#define AP_LCL_AEID    ((AP_TP_ID << 16)|0x08)  /* local aeid               */
#define AP_REM_APT     ((AP_TP_ID << 16)|0x09)  /* remote apt               */
#define AP_REM_APID    ((AP_TP_ID << 16)|0x0a)  /* remote apid              */
#define AP_REM_AEQ     ((AP_TP_ID << 16)|0x0b)  /* remote aeq               */
#define AP_REM_AEID    ((AP_TP_ID << 16)|0x0c)  /* remote aeid              */
#define AP_LCL_TPSUT   ((AP_TP_ID << 16)|0x0d)  /* local TP user title      */
#define AP_REM_TPSUT   ((AP_TP_ID << 16)|0x0e)  /* remote TP user title     */
#define AP_TPFU_AVAIL  ((AP_TP_ID << 16)|0x0f)  /* TP fu's available        */
#define AP_TPFU_SEL    ((AP_TP_ID << 16)|0x10)  /* TP fu's selected         */
#define AP_TP_CATEGORY ((AP_TP_ID << 16)|0x11)  /* primitive categories     */
#define AP_TP_COPYENV  ((AP_TP_ID << 16)|0x12)  /* copy TP env on ind/cnf   */
#define AP_NEXT_AAID   ((AP_TP_ID << 16)|0x13)  /* next aaid                */
#define AP_NEXT_BRID   ((AP_TP_ID << 16)|0x14)  /* next brid                */
#define AP_NEXT_TTNID  ((AP_TP_ID << 16)|0x15)  /* next ttnid               */
#define AP_TP_SEL      ((AP_TP_ID << 16)|0x16)  /* selected TP version      */
#define AP_TP_AVAIL    ((AP_TP_ID << 16)|0x17)  /* available TP version     */
#define AP_TP_STATE    ((AP_TP_ID << 16)|0x18)  /* TP state                 */
#define AP_CONTROL_ID  ((AP_TP_ID << 16)|0x19)  /* Control instance Id.     */
#define AP_URCH        ((AP_TP_ID << 16)|0x1a)  /* Recovery Context Handle  */

/*  States, available from AP_TP_STATE  */

/*
 * The first 25 states map exactly to the 25 states
 * of the OSI TP Service specification
 */

#define AP_TP_UNBOUND                       ((AP_TP_ID << 16) + 0)
#define AP_TP_IDLE                          ((AP_TP_ID << 16) + 1)
#define AP_TP_DATA_XFER                     ((AP_TP_ID << 16) + 2)
#define AP_TP_RECV                          ((AP_TP_ID << 16) + 3)
#define AP_TP_ERROR_RECV                    ((AP_TP_ID << 16) + 4)
#define AP_TP_ERROR                         ((AP_TP_ID << 16) + 5)
#define AP_TP_WHANDcnf                      ((AP_TP_ID << 16) + 6)
#define AP_TP_WHANDrsp                      ((AP_TP_ID << 16) + 7)
#define AP_TP_WHANDrsp_WHANDcnf             ((AP_TP_ID << 16) + 8)
#define AP_TP_WHANDcnf_WENDrsp              ((AP_TP_ID << 16) + 9)
#define AP_TP_WHANDrsp_WENDcnf              ((AP_TP_ID << 16) + 10)
#define AP_TP_WENDcnf                       ((AP_TP_ID << 16) + 11)
#define AP_TP_WENDrsp                       ((AP_TP_ID << 16) + 12)
#define AP_TP_WHAND_GCcnf                   ((AP_TP_ID << 16) + 13)
#define AP_TP_WHAND_GCrsp                   ((AP_TP_ID << 16) + 14)
#define AP_TP_WREADYind                     ((AP_TP_ID << 16) + 15)
#define AP_TP_WREADYind_DATAP               ((AP_TP_ID << 16) + 16)
#define AP_TP_READY                         ((AP_TP_ID << 16) + 17)
#define AP_TP_WPREP_ALLreq                  ((AP_TP_ID << 16) + 18)
#define AP_TP_WPREP_ALLreq_DATAP            ((AP_TP_ID << 16) + 19)
#define AP_TP_WCOMMITind                    ((AP_TP_ID << 16) + 20)
#define AP_TP_COMMIT_WDONEreq               ((AP_TP_ID << 16) + 21)
#define AP_TP_WCOMMIT_COMPind               ((AP_TP_ID << 16) + 22)
#define AP_TP_ROLL_WDONEreq                 ((AP_TP_ID << 16) + 23)
#define AP_TP_WROLL_COMPind                 ((AP_TP_ID << 16) + 24)
#define AP_TP_ZOMBIE                        ((AP_TP_ID << 16) + 25)
```

```
/* The states below are additional to the OSI TP Service standard */

#define AP_TP_PREPARING                      ((AP_TP_ID << 16) + 26)
#define AP_TP_LOGGING_READY                  ((AP_TP_ID << 16) + 27)

/* The states for control of resume and restart */

#define AP_TP_WRESUMEreq                     ((AP_TP_ID << 16) + 28)
#define AP_TP_RESUME                         ((AP_TP_ID << 16) + 29)
#define AP_TP_WRESTARTreq                    ((AP_TP_ID << 16) + 30)
#define AP_TP_RESTART                        ((AP_TP_ID << 16) + 31)
#define AP_TP_WRESTART_COMPLETEind           ((AP_TP_ID << 16) + 32)

/* The states for association allocation */

#define AP_TP_WALLOCcnf                      ((AP_TP_ID << 16) + 33)
#define AP_TP_ALLOCATED                      ((AP_TP_ID << 16) + 34)

/* The state for a log damage node */

#define AP_TP_HEURISTIC_LOG                  ((AP_TP_ID << 16) + 35)

/* Values for AP_TP_FU functional units */

#define AP_TP_POLARIZED_CONTROL     BITL(0)
#define AP_TP_SHARED_CONTROL        BITL(1)
#define AP_TP_COMMIT_AND_CHAINED    BITL(2)
#define AP_TP_COMMIT_AND_UNCHAINED  BITL(3)
#define AP_TP_HANDSHAKE             BITL(4)
#define AP_TP_RECOVERY              BITL(5)

/* Values for AP_TP_CATEGORY */.

#define AP_TP_DIALOGUE  BITL(0)
#define AP_TP_CONTROL   BITL(1)

/* Additional values for AP_MODE */

#define AP_TP_MODE      BITL(2)

/* Constants used in ap_cdata_t */

/* Bits for tp_dialog_env_t mask */

#define AP_AAID_BIT       BITL(0)
#define AP_BRID_BIT       BITL(1)
#define AP_DTNID_BIT      BITL(2)
#define AP_TTNID_BIT      BITL(3)
#define AP_LCL_APT_BIT    BITL(4)
#define AP_LCL_APID_BIT   BITL(5)
#define AP_LCL_AEQ_BIT    BITL(6)
#define AP_LCL_AEID_BIT   BITL(7)
#define AP_LCL_TPSUT_BIT  BITL(8)
#define AP_REM_APT_BIT    BITL(9)
#define AP_REM_APID_BIT   BITL(10)
#define AP_REM_AEQ_BIT    BITL(11)
#define AP_REM_AEID_BIT   BITL(12)
#define AP_REM_TPSUT_BIT  BITL(13)
#define AP_TPFU_SEL_BIT   BITL(14)
```

```
#define AP_TP_SEL_BIT     BITL(15)
#define AP_TP_STATE_BIT   BITL(16)

/* Values for res */

#define AP_TP_ACCEPT            (1)
#define AP_TP_REJ_PROV          (2)
#define AP_TP_REJ_USER          (3)

/* Values for heuristic res, and log record type field */

#define AP_TP_NONE              (0)
#define AP_TP_HEUR_MIX          (1)
#define AP_TP_HEUR_HAZ          (2)

/* Values for log record type field */
#define AP_TP_READY_LOG         (1)
#define AP_TP_COMMIT_LOG        (2)

/* log record flags bit settings */
#define AP_TP_SUPERIOR_SECTION  (1)

/* Values for res_src */

#define AP_TP_SERV_USER         (5)
#define AP_TP_SERV_PROV         (6)
#define AP_APM_SERV_PROV        (7)

/* Values for diag */
#define AP_TP_NRSN              (0)

/* values used for TP_BEGIN_DIALOGUE_CNF  */
#define AP_TP_TPSUT_UNKNOWN     (1)
#define AP_TP_TPSUT_NVAIL_PERM  (2)
#define AP_TP_TPSUT_NVAIL_TRAN  (3)
#define AP_TP_TPSUT_NEEDED      (4)
#define AP_TP_FU_NSUP           (5)
#define AP_TP_FU_COMB_NSUP      (6)
#define AP_TP_ASSOC_RES         (7)
#define AP_TP_RECIPIENT_UNKNOWN (8)

/* values used for APM_ALLOCATE_CNF */
#define AP_TP_CCR_V2_NAVAIL     (1<<0)
#define AP_TP_VER_NAVAIL        (1<<1)
#define AP_TP_CW_REJ            (1<<2)
#define AP_TP_BM_REJ            (1<<3)

/* APM returned values */
/* Values used for APM_ALLOCATE_CNF */
#define AP_TP_ASSOC_NVAIL       (1)
#define AP_TP_BAD_AET           (2)
#define AP_TP_BAD_POOL          (3)
#define AP_TP_POOL_LIMIT        (4)
#define AP_TP_POOL_TIMEOUT      (5)
#define AP_TP_DIALOGUE_REFUSED  (6)
/* Values used for APM_ASSOCIATION_LOST_IND */
#define AP_TP_IN_DIALOGUE       (7)
#define AP_TP_ABORTED           ((AP_TP_ID <<16) | 0)
```

```
/* values used for TP_P_ABORT_IND */
#define AP_TP_PERMANENT         (1)
#define AP_TP_REJ_TRANSACTION   (2)
#define AP_TP_TRANSIENT         (3)
#define AP_TP_PROTOCOL_ERROR    (4)
#define AP_TP_END_CLASH         (5)
#define AP_TP_BEG_END_CLASH     (6)

/* Values for tp_options */

#define AP_TP_TRANSACTION       BITL(0)  /* begin transaction      */
#define AP_TP_CONFIRM           BITL(1)  /* confirmation required  */
#define AP_TP_URGENT            BITL(2)  /* urgency                */
#define AP_TP_PERMITTED         BITL(3)  /* data permitted         */
#define AP_TP_ROLLBACK          BITL(4)  /* rollback               */
#define AP_TP_SYNC_ALLOC        BITL(5)  /* synchronous allocation */
#define AP_TP_ASSOC_ALLOCATED   BITL(6)  /* association allocated  */
#define AP_TP_CONT_WINNER       BITL(7)  /* get contention winner  */
#define AP_TP_SUPERIOR          BITL(8)  /* dialogue to superior   */

typedef struct {
  /* XAP members */
  long              udata_length;    /* length of user-data field          */
  long              rsn;             /* reason for activity/abort/release  */
  long              evt;             /* event that caused abort            */
  long              sync_p_sn;       /* sync point serial number           */
  long              sync_type;       /* sync type                          */
  long              resync_type;     /* resync type                        */
  long              src;             /* source of abort                    */
  long              res;             /* result of primitive                */
  long              res_src;         /* source of result                   */
  long              diag;            /* reason for rejection (if rejected) */
  unsigned long     tokens;          /* tokens identifier                  */
  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-TP members */
  long              tp_options;      /* bit significant TP flags           */
  unsigned long     user_id;         /* abstract syntax or U_ASE identifier*/
  long              tp_fail_count;   /* count of failure conditions        */
  tp_dialog_env_t   *tp_env;         /* dialogue attribute values          */
} ap_tp_cdata_t;

/* Additional error codes for the ap Library Interface */

#define AP_TP_NO_PROVIDER       ERRNO(AP_TP_ID, (0))
#define AP_TP_RESTART_REQD      ERRNO(AP_TP_ID, (1))
#define AP_TP_RESTARTING        ERRNO(AP_TP_ID, (2))
#define AP_TP_BAD_UDATA         ERRNO(AP_TP_ID, (3))
#define AP_TP_BAD_URCH          ERRNO(AP_TP_ID, (5))
#define AP_TP_BAD_CONTROL_ID    ERRNO(AP_TP_ID, (6))
#define AP_TP_BADCD_TP_OPTIONS  ERRNO(AP_TP_ID, (7))
#define AP_TP_BADCD_FAIL_COUNT  ERRNO(AP_TP_ID, (8))
#define AP_TP_BAD_LOG           ERRNO(AP_TP_ID, (9))
#define AP_TP_BAD_TTNID         ERRNO(AP_TP_ID, (10))
#define AP_TP_BAD_NODE          ERRNO(AP_TP_ID, (11))
```

```
/* Macros for encoding and decoding log records */

/* return number of unsigned chars needed to hold a int field */

#define AP_TP_VP_INT_LENGTH(val) \
    (((val) <= 0x7f)? 1: ((val) <= 0x3fff)? 2: 4)

/* return number of unsigned chars needed to hold an variable data value */
#define AP_TP_VP_VAL_LENGTH(len) (len)

/* encode a integer field and return pointer to next free location */

#define AP_TP_VP_ENCODE_INT(ptr, val) \
    (((val) <= 0x7f)? \
        *(ptr)++ = (unsigned char) (val):\
     ((val) <= 0x3fff)? \
        (*(ptr)++ = (unsigned char) (((val) & 0x7f) | 0x80),\
         *(ptr)++ = (unsigned char) ((val) >> 7) ) :\
     (*(ptr)++ = (unsigned char) (((val) & 0x7f) | 0x80),\
      *(ptr)++ = (unsigned char) ((((val) >> 7) & 0x7f) | 0x80),\
      *(ptr)++ = (unsigned char) (((val) >> 14) & 0xff),\
      *(ptr)++ = (unsigned char) (((val) >> 22) & 0xff)), (ptr))

/*
 * encode a variable data field and return pointer to next free location
 */
#define AP_TP_VP_ENCODE_VAL(ptr, len, val) \
    (ptr = ((unsigned char *) memcpy((ptr), (val), (len))) + (len))

/*
 * return the integer field from the current location referenced by ptr
 * and update ptr to reference the next field
 */
#define AP_TP_VP_INT(ptr) \
    (!((ptr)[0] & 0x80))? \
        (unsigned int) *(ptr)++: \
    (!((ptr)[1] & 0x80))? \
        ((ptr) += 2, ((unsigned int) (ptr)[-2] & 0x7f) | \
                      ((ptr)[-1] << 7)): \
    ((ptr) += 4, ((unsigned int) (ptr)[-4] & 0x7f) | \
                 (((ptr)[-3] & 0x7f) << 7) | \
                 ((ptr)[-2] << 14) | \
                 ((ptr)[-1] << 22))

/*
 * return ptr to variable length field and update ptr to reference the
 * next field
 */
#define AP_TP_VP_PTR(ptr, len) \
    ((ptr) += (len), ((ptr) - (len)))
```

Glossary

abstract syntax
The abstract description of a set of data values. Used by an application service element or application to define the data structures to be transferred. Usually expressed using the ASN.1 abstract syntax notation. The Application and Presentation Layers negotiate the set of abstract syntaxes to be used to transfer data values on an association.

ACSE
Association Control Service Elements. OSI Association Control Service Element in the Application Layer of the ISO 7-layer OSI Reference Model. The ISO entity that is responsible for establishing and terminating associations (that is, cooperative relationships) between two applications.

application
An application is a program, typically written by an end-user, designed to achieve some objective.

AE
Application-entity. Defined as the aspects of an application-process pertinent to OSI. An application may contain one or more AEs, each of which performs part of the OSI-related functions required by the application. For example, a network management application might contain one application entity to access an OSI directory service and another to perform the OSI management functions.

An AE communicates with other AEs (using the services of one or more ASEs) to perform its functions.

AET
Application-entity-title. Identifies a particular AE within an application-process. An AET is associated with a single presentation address. An application-entity-title consists of an application-process-title and an application-entity-qualifier \m see ISO 7498-3: 1989, Part 3, Naming and Addressing.

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

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

API
Application Programming Interface. This is a set of services (such as functions in a given programming language) by which the application program communicates with other software components.

ASE
Application Service Element. 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, the connection management service element), whilst others provide services oriented to a particular application (for example, CMISE, the common management information service element).

CRM
Communication Resource Manager. A Resource Manager that offers communication to other Transaction Manager domains.

DTP
In the context of X/Open transaction processing, this refers to distributed transaction processing.

function
A programming language construct, modelled after the mathematical concept. A function encapsulates some behaviour. It is given some arguments as input, performs some processing, and returns some results.

ISO
International Organization for Standardization. A standards organisation with the membership composed of the standards organisations from each participating country. OSI working groups generate the OSI Protocol Suite standards.

interoperability
The ability of software and hardware on multiple machines and from multiple vendors to communicate effectively.

MACF
Multiple Association Control Function.

OSI
Open Systems Interconnection. ISO standard for the interconnection of cooperative (open) computer systems, using the ISO OSI 7-layer Reference Model.

OSI seven-layer model
The ISO Reference Model for OSI (ISO 7498:1984). A conceptual model which provides a common basis for describing communications protocols and services.

pdu
Protocol Data Unit. The data unit exchanged by peer protocol entities. Examples are apdu (Application Layer), ppdu (Presentation Layer) and spdu (Session Layer).

portability
Machine-independent. Applied to software that can be readily ported to different machines.

presentation context
An association of an *abstract syntax* with a *transfer syntax*, negotiated by the Presentation Layer when an application association is established.

The Application Layers propose the abstract syntaxes to be used on the association; the Presentation Layer negotiates the transfer syntaxes to be used to support each of the abstract syntaxes. When transferring data, the Application Layer identifies the presentation context to be used to encode and decode the data.

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

primitive
An event that occurs at an interface between the user of the service and the service provider in an open system.

protocol
A specification for an agreed procedure to enable exchange of information between cooperating entities, via interfaces which provide the necessary capability to cover format of messages, data checks, flow control and error handling.

A set of protocols governing the exchange of information between remote systems, and set of interfaces covering the exchange between adjacent protocol levels, are collectively referred to as a protocol hierarchy or protocol stack.

RPC
Remote Procedure Call. A call by one endpoint in a communications link for the other endpoint to perform a procedure.

RM
Resource Manager. Manages a specified part of a computer's shared resources (for example, a database) in a transaction processing system.

RTI-SUI
Remote Task Invocation - Service User Invocation. A server that uses the RTI service provided by the RTI Protocol Machine.

SACF
Single Association Control Function.

Session Layer
Fifth layer in the OSI Basic Reference Model. This layer provides services that allow AEs to organise and synchronise their interactions. In addition to the connection and data transfer services of the Transport Layer, the Session Layer provides orderly release, synchronisation, activity management and half-duplex operation.

SQL
Structured Query Language. The X/Open SQL definition is based on ISO 9075:1987 and the identical American National Standard Database Language SQL ANS X3.135-1986, with variances to meet a spectrum of existing UNIX implementations and extend capability to reflect common usage.

transaction
A transaction is a discrete unit of work which is characterised by four basic properities known as the ACID properties. The acronym ACID stands for Atomicity, Consistency, Isolation and Durability. These properities express that either all or none of the operations of a transaction are performed, that intermediate results of a transaction are not visible to other transactions, which are executed at the same time, and that all effects of a completed transaction are permanent.

TM
Transaction Manager. Manages global transactions in a transaction processing system, including coordinating the decision to commit them or roll them back.

TP
Transaction Processing. Operations in a data processing system, in which transactions are processed to completion as they arise.

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

Transport Layer
Fourth layer in the OSI Basic Reference Model. This layer provides a transparent connection and duplex data transfer service between OSI end systems. Transport Layer functions include end-to-end sequencing, flow control, error detection and recovery.

TxRPC
Transactional RPC. An RPC initiated from within the scope of a transaction.

XA
The name given to the interface between Transaction Manager and Resource Manager, in a transaction processing system. It lets the TM structure the work of RMs into global transactions and coordinate global transaction completion and recovery.

XAP
X/Open ACSE/Presentation (XAP) application programming interface.

XAP-TP
X/Open ACSE/Presentation (XAP) programming interface, with transaction processing extension (XAP-TP).

XTP
X/Open Distributed Transaction Processing Model.

Index