Technical Standard

Distributed Transaction Processing: The XATMI Specification

[This page intentionally left blank]

X/Open CAE Specification

Distributed Transaction Processing:

The XATMI Specification

X/Open Company Ltd.

X/Open CAE Specification

Distributed Transaction Processing: The XATMI Specification

ISBN: 1-85912-130-6

X/Open Document Number: C506

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 XATMI interface, which is an application program interface to a Communication Resource Manager (CRM). The XATMI interface allows an application program to communicate with other application programs using a client-server paradigm and optionally to include those other application programs in a global transaction. It also defines the Application Service Element (ASE) for XATMI.

The structure of this specification is as follows:

- Part 1: XATMI Communication Application Programming Interface (API)
  - Chapter 1 is an introduction to the XATMI API.
  - Chapter 2 provides an introduction to the X/Open DTP model and fundamental definitions for the API.
  - Chapter 3 is an overview of the C-language bindings for the XATMI API.
  - Chapter 4 contains C-language header file information for the XATMI API.
  - Chapter 5 contains C reference manual pages for each routine in the XATMI API.
  - Chapter 6 is an overview of the COBOL language bindings for the XATMI API.
  - Chapter 7 contains COBOL reference manual pages for each routine in the XATMI API.
  - Chapter 8 contains state tables describing the legal sequences in which calls to the XATMI API can be made.
  - Chapter 9 describes the C typed buffers and COBOL typed records that must be supported by an XATMI implementation.
- Part 2: XATMI Application Service Element (ASE)
  - Chapter 10 describes the mapping of the communication model used by XATMI to the OSI TP Communication Model.
  - Chapter 11 contains the definition of the XATMI Application Context.
  - Chapter 12 contains the definition of the XATMI-ASE service primitives.

- — Chapter 13 describes the protocol procedures for the XATMI-ASE.
- — Chapter 14 defines the structure and encoding of the XATMI-ASE Application Protocol Data Units (APDUs).

- Part 3: API Appendices
  - — Appendix A contains examples written in C.
  - — Appendix B contains examples written in COBOL.
  - — Appendix C describes the necessary extensions to the TX interface.
- Part 4: ASE Appendix
  - — Appendix D contains several examples of the usage of the XATMI-ASE.

There is an index at the end.

Intended Audience

Parts 1 and 3 of this document are intended for application programmers who wish to write portable programs that use global transactions. The whole document is of interest to implementors of the XATMI application programming interface.

All readers are expected to be familiar with the X/Open documents **Distributed Transaction Processing Reference Model** and **Distributed Transaction Processing: The TX (Transaction Demarcation) Specification**. Implementors are also expected to be familiar with the X/Open document **Distributed Transaction Processing: The XA Specification** and the ISO Open Systems Interconnection (OSI) standards listed in **Referenced Documents** on page xvii.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for 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:
  - — variable names, for example, substitutable argument prototypes and environment variables
  - — C-language functions; these are shown as follows: *name*()
- Normal font is used for the names of constants and literals. COBOL function names are also shown in normal font.
- The notation **<file.h>** indicates a C-language header file.
- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values, which may be declared in appropriate C-language header files by means of the C **#define** construct.
- The notation [ABCD] is used to identify a coded return value in C, or the value set in COBOL.
- Syntax and code examples are shown in `fixed width` font.
- Variables within syntax statements are shown in *`italic fixed width`* font.

Note: Syntax statements use the same typographical conventions for C and COBOL. Therefore COBOL syntax statements deviate from the referenced COBOL standard in the following ways:

- No underlining is used with mandatory elements.
- No options are shown; for other valid formats see the X/Open **COBOL Language** specification.
- Substitutable names are shown in italics.

Trade Marks

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

Referenced Documents

The following standards are referenced in this specification:

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

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).

ISO C
: ISO / IEC 9899: 1990, Programming Languages — C (technically identical to ANSI standard X3.159-1989).

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

ISO 8650
: ISO 8650: 1988, Information Processing Systems — Open Systems Interconnection — Protocol specification for the Association Control Service Element.

ISO / IEC 9545
: ISO / IEC 9545: 1989, Information Technology — Open Systems Interconnection — Application Layer Structure.

ISO / IEC 9804
: ISO / IEC 9804: 1994, Information Technology — Open Systems Interconnection — Service Definition for the Commitment, Concurrency, and Recovery Service Element.

ISO / IEC 9805
: ISO / IEC 9805: 1994, Information Technology — Open Systems Interconnection — Protocol Specification for the Commitment, Concurrency, and Recovery Service Element.

OSI TP
: ISO / IEC 10026, Information Technology — Open Systems Interconnection — Distributed Transaction Processing, Parts 1 to 5:

    Part 1: 1992, OSI TP Model
    Part 2: 1992, OSI TP Service
    Part 3: 1992, Protocol Specification
    Part 4: 1995, Protocol Implementation Conformance Statement (PICS) proforma
    Part 5: DIS 1993, Application context proforma and guidelines when using OSI TP.

OSI TP Profiles
: ISO / IEC ISP 12061: 1995, Information Technology — Open Systems Interconnection — International Standardized Profiles: OSI Distributed Transaction Processing, Parts 5, 7 and 9:

    Part 5: Application supported transactions — Polarized control (ATP11)
    Part 7: Provider supported unchained transactions — Polarized control (ATP21)
    Part 9: Provider supported chained transactions — Polarized control (ATP31).

The following X/Open documents are referenced in this specification:

COBOL
: X/Open CAE Specification, December 1991, COBOL Language (ISBN: 1-872630-09-X, C192 or XO/CAE/91/200).

CPI-C, Version 2
: X/Open CAE Specification, November 1995, Distributed Transaction Processing: The CPI-C Specification, Version 2 (ISBN: 1-85912-135-7, C419).

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

TX
: X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN: 1-85912-094-6, C504).

TxRPC
: X/Open CAE Specification, October 1995, Distributed Transaction Processing: The TxRPC Specification (ISBN: 1-85912-115-2, C505).

XA
: X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN: 1-872630-24-3, C193 or XO/CAE/91/300).

XA+
: X/Open Snapshot, July 1994, Distributed Transaction Processing: The XA+ Specification, Version 2 (ISBN: 1-85912-046-6, S423).

XAP-TP
: X/Open CAE Specification, April 1995, ACSE/Presentation: Transaction Processing API (XAP-TP) (ISBN: 1-85912-091-1, C409).

X/Open CAE Specification

Part 1:

XATMI Communication Application Programming Interface (API)

X/Open Company Ltd.

Chapter 1

Introduction

This chapter provides an outline of the X/Open Distributed Transaction Processing (DTP) model and explains the position of this specification as one of the Communication Resource Manager (CRM) interfaces.

1.1 X/Open DTP Model

The X/Open Distributed Transaction Processing (DTP) model is a software architecture that allows multiple application programs to share resources provided by multiple resource managers, and allows their work to be coordinated into global transactions.

The X/Open DTP model comprises five basic functional components:

- an Application Program (AP), which defines transaction boundaries and specifies actions that constitute a transaction
- Resource Managers (RMs) such as databases or file access systems, which provide access to resources
- a Transaction Manager (TM), which assigns identifiers to transactions, monitors their progress, and takes responsibility for transaction completion and for coordinating failure recovery
- Communication Resource Managers (CRMs), which control communication between distributed applications within or across TM domains
- a communication protocol, which provides the underlying communication services used by distributed applications and supported by CRMs.

X/Open DTP publications based on this model specify portable Application Programming Interfaces (APIs) and system-level interfaces that facilitate:

- portability of application program source code to any X/Open environment that offers those APIs
- interchangeability of TMs, RMs and CRMs from various sources
- interoperability of diverse TMs, RMs and CRMs in the same global transaction.

Chapter 2 defines each component in more detail and illustrates the flow of control.

1.2 X/Open Communication Resource Manager Interfaces

An important aspect of distributed transaction processing applications is communication. Within the product domain for DTP tools, there are several popular communication paradigms in common use today or expected to be in common use in the future. The communication paradigm chosen can significantly influence the architecture of the application. The unique strengths of each paradigm make it attractive for specific applications.

The referenced **DTP** guide defines a functional component known as a Communication Resource Manager (CRM), which provides access to a communication medium between applications.

Because it is not possible to choose a single communication paradigm applicable to the entire broad range of DTP applications, X/Open provides application programming interfaces (APIs) for the most popular paradigms in order to bring the benefits of open systems to the widest possible range of transaction processing applications. These are the request/response paradigm and the conversational paradigm.

Many applications already running on open systems use the request/response paradigm. X/Open specifications for this paradigm are the library-based XATMI CRM interface (see this document) and the IDL-based TxRPC CRM interface (see the referenced **TxRPC** specification). TxRPC fits within the context of the X/Open Distributed Computing Services Framework (XDCS) and allows application writers to invoke remote procedure calls (RPCs) in the same form as local procedures, but with transaction semantics.

For applications choosing to use the conversational paradigm, where communication takes place through an application-defined exchange of messages, X/Open offers the library-based interfaces XATMI (see this document) and CPI-C (see the referenced **CPI-C** specification).

Chapter 2

Model and Definitions

This chapter discusses the XATMI interface in general terms and provides necessary background material for the rest of the specification. The chapter shows the relationship of the interface to the X/Open DTP model. The chapter also states the design assumptions that the interface uses and shows how the interface addresses common DTP concepts.

2.1 X/Open DTP Model

The boxes in the figure below are the functional components and the connecting lines are the interfaces between them. The arrows indicate the directions in which control may flow.

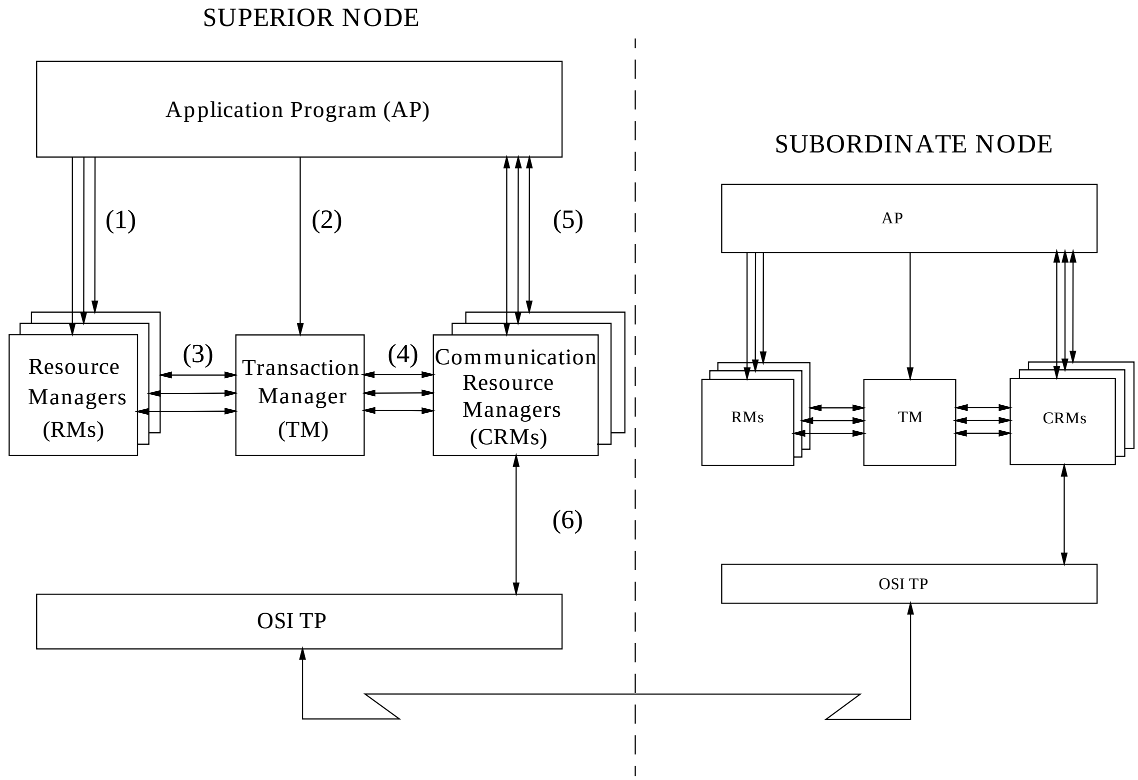


Figure 2-1 Functional Components and Interfaces

Descriptions of the functional components shown can be found in Section 2.1.1 on page 6. The numbers in brackets in the above figure represent the different X/Open interfaces that are used in the model. They are described in Section 2.1.2 on page 7.

For more details of this model and diagram, including detailed definitions of each component, see the referenced **DTP** guide.

2.1.1 Functional Components

Application Program (AP)

The application program (AP) implements the desired function of the end-user enterprise. Each AP specifies a sequence of operations that involves resources such as databases. An AP defines the start and end of global transactions, accesses resources within transaction boundaries, and normally makes the decision whether to commit or roll back each transaction.

Where two or more APs cooperate within a global transaction, the X/Open DTP model supports three *paradigms* for AP to AP communication. These are the TxRPC, XATMI and CPI-C interfaces.

Transaction Manager (TM)

The transaction manager (TM) manages global transactions and coordinates the decision to start them, and commit them or roll them back. This ensures atomic transaction completion. The TM also coordinates recovery activities of the resource managers when necessary, such as after a component fails.

Resource Manager (RM)

The resource manager (RM) manages a defined part of the computer's shared resources. These may be accessed using services that the RM provides. Examples for RMs are database management systems (DBMSs), a file access method such as X/Open ISAM, and a print server.

In the X/Open DTP model, RMs structure all changes to the resources they manage as recoverable and atomic transactions. They let the TM coordinate completion of these transactions atomically with work done by other RMs.

Communication Resource Manager (CRM)

A CRM allows an instance of the model to access another instance either inside or outside the current TM Domain. Within the X/Open DTP model, CRMs use OSI TP services to provide a communication layer across TM Domains. CRMs aid global transactions by supporting the following interfaces:

- the communication paradigm (TxRPC, XATMI or CPI-C) used between an AP and CRM
- XA+ communication between a TM and CRM
- XAP-TP communication between a CRM and OSI TP.

A CRM may support more than one type of communication paradigm, or a TM Domain may use different CRMs to support different paradigms. The XA+ interface provides global transaction information across different instances and TM Domains. The CRM allows a global transaction to extend to another TM Domain, and allows TMs to coordinate global transaction commit and abort requests from (usually) the superior AP. Using the above interfaces, information flows from superior to subordinate and *vice versa*.

2.1.2 Interfaces between Functional Components

There are six interfaces between software components in the X/Open DTP model. The numbers correspond to the numbers in Figure 2-1 on page 5.

(1) **AP-RM.** The AP-RM interfaces give the AP access to resources. X/Open interfaces, such as SQL and ISAM, provide AP portability. The X/Open DTP model imposes few constraints on native RM APIs. The constraints involve only those native RM interfaces that define transactions. (See the referenced **XA** specification.)

(2) **AP-TM.** The AP-TM interface (the TX interface) provides the AP with an Application Programming Interface (API) by which the AP coordinates global transaction management with the TM. For example, when the AP calls *tx_begin*( ) the TM informs the participating RMs of the start of a global transaction. After each request is completed, the TM provides a return value to the AP reporting back the success or otherwise of the TX call.

For details of the AP-TM interface, see the referenced **TX** (Transaction Demarcation) specification.

(3) **TM-RM.** The TM-RM interface (the XA interface) lets the TM structure the work of RMs into global transactions and coordinate completion or recovery. The XA interface is the bidirectional interface between the TM and RM.

The functions that each RM provides for the TM are called the *xa_**( ) functions. For example the TM calls *xa_start*( ) in each participating RM to start an RM-internal transaction as part of a new global transaction. Later, the TM may call in sequence *xa_end*( ) *xa_prepare*( ) and *xa_commit*( ) to coordinate a (successful in this case) two-phase commit protocol. The functions that the TM provides for each RM are called the *ax_**( ) functions. For example an RM calls *ax_reg*( ) to register dynamically with the TM.

For details of the TM-RM interface, see the referenced **XA** specification.

(4) **TM-CRM.** The TM-CRM interface (the XA+ interface) supports global transaction information flow across TM Domains. In particular TMs can instruct CRMs by use of *xa_**( ) function calls to suspend or complete transaction branches, and to propagate global transaction commitment protocols to other transaction branches. CRMs pass information to TMs in subordinate branches by use of *ax_**( ) function calls. CRMs also use *ax_**( ) function calls to request the TM to create subordinate transaction branches, to save and retrieve recovery information, and to inform the TM of the start and end of blocking conditions.

For details of the TM-CRM interface, see the referenced **XA+** specification.

The XA+ interface is a superset of the XA interface and supersedes its purpose. Since the XA+ interface is invisible to the AP, the TM and CRM may use other methods to interconnect without affecting application portability.

(5) **AP-CRM.** X/Open provides portable APIs for DTP communication between APs within a global transaction. The API chosen can significantly influence (and may indeed be fundamental to) the whole architecture of the application. For this reason, these APIs are frequently referred to in this specification and elsewhere as *communication paradigms*. In practice, each paradigm has unique strengths, so X/Open offers the following popular paradigms:

- the TxRPC interface (see the referenced **TxRPC** specification)
- the XATMI interface (see this document)
- the CPI-C interface (see the referenced **CPI-C** specification).

X/Open interfaces, such as the three CRM APIs listed above, provide application portability across products offering the same CRM API. The X/Open DTP model imposes few constraints on native CRM APIs.

(6) **CRM-OSI TP.** This interface (the XAP-TP interface) provides a programming interface between a CRM and Open Systems Interconnection Distributed Transaction Processing (OSI TP) services. XAP-TP interfaces with the OSI TP Service and the Presentation Layer of the seven-layer OSI model. X/Open has defined this interface to support portable implementations of application-specific OSI services. The use of OSI TP is mandatory for communication between heterogeneous TM domains. For details of this interface, see the referenced **XAP-TP** specification and OSI TP standards.

2.2 Definitions

For additional definitions see the referenced **DTP** guide.

2.2.1 Transaction

A transaction is a complete unit of work. It may comprise many computational tasks, which may include user interface, data retrieval, and communication. A typical transaction modifies shared resources. (The OSI TP standards (model) defines transactions more precisely.)

Transactions must be able to be *rolled back*. A human user may roll back the transaction in response to a real-world event, such as a customer decision. A program can elect to roll back a transaction. For example, account number verification may fail or the account may fail a test of its balance. Transactions also roll back if a component of the system fails, keeping it from retrieving, communicating, or storing data. Every DTP software component subject to transaction control must be able to undo its work in a transaction at any time that it is rolled back.

When the system determines that a transaction can complete without failure of any kind, it *commits* the transaction. This means that changes to shared resources take permanent effect. Either commitment or rollback results in a consistent state. *Completion* means either commitment or rollback.

2.2.2 Transaction Properties

Transactions typically exhibit the following properties:

Atomicity The results of the transaction's execution are either all committed or all rolled back.

Consistency A completed transaction transforms a shared resource from one valid state to another valid state.

Isolation Changes to shared resources that a transaction effects do not become visible outside the transaction until the transaction commits.

Durability The changes that result from transaction commitment survive subsequent system or media failures.

These properties are known by their initials as the **ACID** properties. In the X/Open DTP model, the TM coordinates Atomicity at global level whilst each RM is responsible for the Atomicity, Consistency, Isolation and Durability of its resources.

2.2.3 Distributed Transaction Processing

Within the scope of this document, DTP systems are those where work in support of a single transaction may occur across RMs. This has several implications:

- The system must have a way to refer to a transaction that encompasses all work done anywhere in the system.
- The decision to commit or roll back a transaction must consider the status of work done anywhere on behalf of the transaction. The decision must have uniform effect throughout the DTP system.

Even though an RM may have an X/Open-compliant interface such as Structured Query Language (SQL), it must also address these two items to be useful in the DTP environment.

2.2.4 Global Transactions

Every RM in the DTP environment must support transactions as described in Section 2.2.1 on page 9. Many RMs already structure their work into recoverable units.

In the DTP environment, many RMs may operate in support of the same unit of work. This unit of work is a *global transaction*. For example, an AP might request updates to several different databases. Work occurring anywhere in the system must be committed atomically. Each RM must let the TM coordinate the RM's recoverable units of work that are part of a global transaction.

Commitment of an RM's internal work depends not only on whether its own operations can succeed, but also on operations occurring at other RMs, perhaps remotely. If any operation fails anywhere, every participating RM must roll back all operations it did on behalf of the global transaction. A given RM is typically unaware of the work that other RMs are doing. A TM informs each RM of the existence, and directs the completion, of global transactions. An RM is responsible for mapping its recoverable units of work to the global transaction.

2.2.5 Transaction Branches

A global transaction has one or more *transaction branches* (or *branches*). A branch is a part of the work in support of a global transaction for which the TM and the RM engage in a separate but coordinated transaction commitment protocol. Each of the RM's internal units of work in support of a global transaction is part of exactly one branch.

A global transaction might have more than one branch when, for example, the AP uses a CRM to communicate with remote applications. The CRM asks the TM to create a new transaction branch prior to accessing a remote AP for the first time. Subsequent accesses to the same remote AP are typically done within the same transaction branch. Accesses to different remote APs are typically done in separate transaction branches.

After the TM begins the transaction commitment protocol, the RM receives no additional work to do on that transaction branch. The RM may receive additional work on behalf of the same transaction, from different branches. The different branches are related in that they must be completed atomically. However, the TM directs the commitment protocol for each branch separately. That is, an RM receives a separate commitment request for each branch.

2.2.6 Clients, Servers and Services

The XATMI interface embodies a programming model whereby application programs are structured either as clients or as servers. A *client* is an AP that requests services to be performed. The structure of a client AP is defined entirely by the application writer.

A *service* is an AP that performs a specific application function on behalf of clients. The structure of a service routine, that is the mechanism by which a service is invoked and terminated, is defined by the XATMI interface.

There are two types of service:

- *Request/response services* receive a single request and produce at most a single response to the request. The *request* is the application data sent from the client to the service. The service processes the request and returns application data to the client by means of at most one *response*.
- *Conversational services* are invoked by means of a *connection request* from the client. Once the connection is established and the service invoked, the client and the service can exchange data in an application-specific manner until the service returns, whereupon the connection is logically terminated.

A service may itself invoke another service. In this case the first service acts like a client. The term *requester* is used to refer to any AP that invokes a service, whether that AP is itself a service or a client.

A *server* is an entity that dispatches a service to satisfy a client's request. A server may offer one or more distinct services while a particular service may be offered by one or more servers. The mechanism for incorporating services into servers is defined by the CRM software implementing the XATMI interface.

2.2.7 Application-level Chaining

The **TX** (Transaction Demarcation) specification allows applications to specify the use of chained transactions. With chained transactions, an application explicitly indicates the start of the first transaction. Thereafter, completion of one transaction automatically starts another one.

When using unchained transactions, an application must explicitly start each transaction. A new transaction does not implicitly start when the application completes a previous transaction. Unchained transactions allow an application to perform operations outside global transactions.

Service routines are either invoked in a global transaction or outside a global transaction. If as part of its transaction a requester invokes a service routine, the service can participate in only that transaction and the service does not call any transaction demarcation functions. If the requester invokes a service routine outside any transaction, the service routine can originate and complete any number of transactions using the TX (Transaction Demarcation) interface.

2.2.8 Local Configuration

The administrator specifies the location of (the means of gaining access to) all APs that can be named as services. This document uses the term *local configuration* to refer to the complete set of information on such APs. An AP requesting communication identifies the desired service by a symbolic name. This symbolic name is mapped (in an implementation-specific way) to a particular service routine based on local configuration information, run-time tables based on that information or both.

2.3 Design Principles

2.3.1 General Principles

The Client-Server CRM interface adheres to these general principles:

- The interface isolates application programming from its environment. For example, the XATMI interface insulates application programmers from lower-level communication and networking protocols. The programmer deals with a small number of well-defined communication methods that naturally support transaction-based communication. Instantiation, structure, and management of the associated resources are implementation-defined and outside the scope of the XATMI interface.
- The interface includes functionality that can be mapped to and from the OSI TP protocol.
- The interface allows APs to use location-transparent naming rather than physical locations. For example, requesters can ask for the DEBIT service without having to worry about how many such services are available or where they are located.
- The interface allows APs to pass application data without regard for machine boundaries or processor architectures. That is, the XATMI interface includes mechanisms for transparently encoding and decoding application data across heterogeneous processor types.

2.3.2 Relationship to OSI TP

X/Open assumes that communication between heterogeneous TM domains uses CRMs that follow the transaction management protocol specified in the OSI TP standards. Either proprietary protocols or OSI TP could be used between homogeneous TM domains.

The XATMI interface is designed to be mappable to and from the OSI TP protocol. It is a goal that there should be a way, using OSI TP, to convey the result of any permitted use of the XATMI interface.

Chapter 3

C-language Interface Overview

This chapter gives an overview of the XATMI interface and describes its relationship to the TX interface. In an X/Open DTP system, XATMI is the interface between an AP and a CRM, and TX is the interface between an AP and a TM.

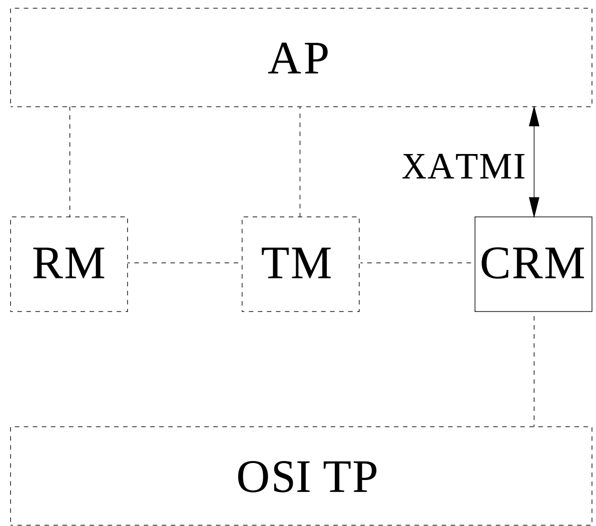


Figure 3-1 The XATMI Interface

The XATMI interface is the API to a CRM that supports a client-server paradigm in an X/Open DTP system. This interface offers the following programming models (see also the definitions in Chapter 2):

- The request/response service paradigm allows the writing of a structured service AP routine that receives a single request and may produce a single reply. The CRM automatically initialises the communication path to the server and automatically invokes the AP service routine.
- The conversational service paradigm provides for the same automatic setup as for the request/response service paradigm, but lets the AP service routine exchange data with the requester multiple times and in an application-defined sequence.

The CRM must know (typically from the local configuration) which paradigm is followed by the AP routine addressed by any given request for communication, because it must enforce a different state table in each case.

This chapter gives an overview of the C-language interface; it describes each paradigm: its attributes, the XATMI functions available in each paradigm and their usage, and programming examples. This chapter also explains the concept of typed buffers. Chapter 5 contains reference manual pages for each routine in alphabetical order.

3.1 Index to Functions in the XATMI Interface

Name	Description	See
	Typed Buffer Functions	
tpalloc	Allocate a typed buffer.	Section 3.2 on page 15.
tpfree	Free a typed buffer.	Section 3.2 on page 15.
tprealloc	Change the size of a typed buffer.	Section 3.2 on page 15.
tptypes	Determine information about a typed buffer.	Section 3.2 on page 15.
	Functions for Writing Service Routines	
tpservice	Template for service routines.	Section 3.3 on page 15.
tpreturn	Return from a service routine.	Section 3.3 on page 15.
	Functions For Dynamically Advertising Service Names	
tpadvertise	Advertise a service name.	Section 3.4 on page 16.
tpunadvertise	Unadvertise a service name.	Section 3.4 on page 16.
	Functions for request/response Services	
tpacall	Send a service request.	Section 3.5 on page 17.
tpcall	Send a service request and synchronously await its reply.	Section 3.5 on page 17.
tpcancel	Cancel a call descriptor for an outstanding reply.	Section 3.5 on page 17.
tpgetrply	Get a reply from a previous service request.	Section 3.5 on page 17.
	Functions for Conversational Services	
tpconnect	Establish a conversational service connection.	Section 3.6 on page 18.
tpdiscon	Terminate a conversational service connection abortively.	Section 3.6 on page 18.
tprecv	Receive a message in a conversational connection.	Section 3.6 on page 18.
tpsend	Send a message in a conversational connection.	Section 3.6 on page 18.

Table 3-1 C-Language XATMI Functions

The *tp**() routines are the application interface provided by X/Open-compliant CRMs implementing the XATMI interface. An AP may call these routines.

An AP must call the *tp**() routines in accordance with the state tables in Chapter 8. However, if an AP calls more than one CRM, or has more than one outstanding request or conversational connection using an XATMI CRM, its calls to each do not depend on the state of its dealings with any other RM, specific request or connection.

3.2 Typed Buffers

In order to send data to another AP, the sending AP first places the data in a *buffer*. The XATMI interface supports *typed buffers*. A typed buffer contains data and has associated with it a type and possibly a subtype, that indicate the meaning or interpretation of the data. A combination of type and subtype corresponds to a host-language structure definition. Typed buffers are specified via character strings, but actual types and subtypes are defined in an implementation-specific manner.

Typed buffers are dynamically allocated. In addition, their size, type and subtype are allowed to change on receipt of a buffer that is either larger or of a different type and subtype from the original buffer. An AP calls *tpalloc*( ) to allocate a typed buffer of a specified type and subtype, can call *tprealloc*( ) to increase its size, and must eventually call *tpfree*( ) to dispose of it. A receiver of a typed buffer can call *tptypes*( ) to determine the type and subtype of a buffer as well as its size.

X/Open predefines three buffer types for the C XATMI interface that all implementations support (see Chapter 9). An AP can specify by the type and subtype that a buffer's structure is interpreted by the AP.

3.3 Service Paradigm

The *service paradigm* refers to the common aspect of automatic setup and invocation in both the request/response and conversational service paradigms.

Service routines are coded as C-language functions. A service routine is invoked from implementation-specific dispatching code contained within a server. Handling of the communication path is independent of the service and is the responsibility of the CRM. From an application writer's viewpoint, communication between a requester and a service routine is utilised only for the duration of the function invocation.

The *tpservice*( ) reference manual page presents a standard form for coding a service. The arguments to the function are set by the dispatching code at the server location based on the service request received from the requester.

tpservice( ) is the template for writing service routines. This template is used both for services that receive requests via *tpcall*( ) or *tpacall*( ) routines, and services that communicate via *tpconnect*( ), *tpsend*( ) and *tprecv*( ) routines.

tpreturn( ) is used to send a service's reply message. If an AP receiving the reply is waiting in either *tpcall*( ), *tpgetrply*( ) or *tprecv*( ), then after a successful call to *tpreturn*( ), the reply is available in the receiving AP's buffer.

Services can accept more than one kind of typed buffer. In fact, services can accept one buffer type on input and send a different buffer type in the response. The buffer types that a service accepts can be specified in the local configuration.

3.4 Service Names and Dynamic Advertising

The requester identifies a service with which it wishes to communicate in the service name parameter to *tpacall*( ), *tpcall*( ) or *tpconnect*( ). This parameter is a character string (for example, "DEBIT" or "CREDIT") and is completely defined by the application.

When servers are started, they *advertise* the set of services that they offer (in an implementation-specific manner). At run time, service routines themselves can alter a server's set of service advertisements. AP service routines may choose to do this, for example, based upon time of day or information received as part of a service request.

tpadvertise( ) allows a server to advertise a new service that it offers. The function takes two parameters: the service name and the actual C language routine that should be invoked when a request for the service name is received by the server. Since the service name may differ from the routine name, different service names can be mapped to the same function.

tpunadvertise( ) allows a server to unadvertise a service that it offers. Even though a particular service may be unadvertised by one server, it may still be offered by others.

Information about service names may be kept in the local configuration. Because each service supports either the request/response or the conversational service paradigm, the local configuration may contain information labeling each service name appropriately.

3.5 Request/Response Service Paradigm

Requests can be issued to services in two ways: synchronously or asynchronously. In both methods, the requester can state whether the request should be sent as part of the caller's current transaction.

3.5.1 Synchronous Request/Response

The *tpcall*( ) function sends a request to the specified service, and returns any response in an application-defined typed buffer. The call to *tpcall*( ) returns after any expected response arrives.

3.5.2 Asynchronous Request/Response

The *tpacall*( ) function also sends a request to the specified service, but it returns without waiting for the service's response, letting the requester do additional work while the service routine processes its request. Using the *tpacall*( ) function allows a requester to exploit parallelism within an application since multiple requests can be simultaneously processed. The *tpacall*( ) function returns to its caller a *call descriptor* that is used by the requester to eventually get its reply. If the requester does not require any reply, the requester must indicate that a reply is not expected. However, in this particular case, the request must not be issued in transaction mode.

The *tpgetrply*( ) function waits to receive a service reply corresponding to a specified request. The function returns the response in an application-defined typed buffer.

A requester not wanting to receive a reply from a previously sent request can call the *tpcancel*( ) function. This function informs the CRM that any response should be silently discarded. The *tpcancel*( ) function does not prevent the service from completing; rather, it relieves the requester from having to receive an unwanted response. It is an error to attempt to cancel a call descriptor associated with a global transaction.

3.5.3 Programming Example

See Appendix A for an example of request/response programming in the C programming language.

3.6 Conversational Service Paradigm

In this paradigm, a requester invokes a service routine and converses with it in an application-defined manner. Thus, several messages can be exchanged before the service routine returns ending the conversation. The conversation takes place in a *half-duplex* manner. That is, only one program can send data at a time. Also, the receiver cannot send data until the sender yields its control of the conversation.

The requester initiates conversational communication with a service by calling the *tpconnect*( ) function. This function optionally passes application data to the service and specifies which program initially has control of the connection. The requester is returned a descriptor that it uses to refer to the newly established connection during subsequent communication. The functions *tpsend*( ) and *tprecv*( ) allow APs to exchange data over an open connection.

On the server side of the connection, the CRM listens for and accepts the incoming connection request. The service routine matching the requester's named service is dispatched along with a descriptor that refers to the connection as well as any application data sent as part of the requester's call to *tpconnect*( ).

A conversational service's communication path with its requester is terminated by the CRM in an orderly manner after the service returns by calling *tpreturn*( ). If the requester wishes to terminate the conversation abortively, rather than orderly, it can call *tpdiscon*( ). This function terminates a connection in a manner that data in transit may be lost and any active transaction associated with that connection is rolled back.

Because communication in the conversational service paradigm is ''longer lived'' than that of the request/response paradigm, communication *events* that occur during the course of a conversation are reported to either the requester, the service, or both as appropriate. For example, the AP that controls the connection yields control to the receiver by sending it an event. Other events include orderly as well as abortive connection termination.

3.6.1 Programming Example

See Appendix A for an example of conversational programming in the C programming language.

3.7 Transaction Implications

The XATMI interface relies on the **TX** (Transaction Demarcation) interface, published separately, for global transaction demarcation and management. In addition, certain functions in the XATMI interface directly affect the progress of a global transaction.

3.7.1 Transaction Functions Affecting the XATMI Interface

Demarcation

The XATMI interface relies on the following functions of the Transaction Demarcation (TX) interface:

tx_begin( ) A demarcation function that indicates that subsequent work performed by the calling AP is in support of a global transaction.

tx_commit( ) A demarcation function that commits all work done on behalf of the current global transaction.

tx_rollback( ) A demarcation function that rolls back all work done on behalf of the current global transaction.

The effect of the TX functions on this specification is that an AP detects that the partner's TM has requested completion of the transaction by means of return codes, communication events or errors. The AP may use this information to instruct its TM and its subordinates on how to complete the transaction.

As described in Section 2.2.6 on page 10, APs may generate both request/response and conversational requests. XATMI, by default, includes such requests within the global transaction if one is active at the time the requests are initiated. XATMI allows an AP to establish communication requests outside the boundaries of the global transaction through flags available on the API. Additionally, communication requests established before the global transaction is begun are also not included in the global transaction. The state and validity of these non-transactional requests are not affected by the transaction demarcation (TX) functions. Non-transactional descriptors may be affected with respect to timeout as described below; however, they are not invalidated by any transaction-related timeouts.

As described above, both request/response and conversational requests generated by the AP are, by default, included in a global transaction if one is active. The descriptors relating to these communications should be closed, that is terminated normally as described in the reference manual pages, prior to invocation of *tx_commit*( ) or *tx_rollback*( ). If such descriptors are active, that is not closed, at the time *tx_commit*( ) or *tx_rollback*( ) is invoked, then the descriptors are invalidated by the TM and the transaction is rolled back. Note that transaction chaining as defined by the transaction demarcation (TX) functions is allowed even though transaction-related XATMI communication descriptors do not survive transaction boundaries.

Service routines as defined in XATMI may be invoked in transaction mode. In that case, they are subject to the following characteristics with respect to transaction demarcation: *tx_begin*( ) fails with a protocol error because the service routine is already in a transaction; *tx_commit*( ) and *tx_rollback*( ) fail with a protocol error because they are not the originators of the transaction.

Timeouts

The timeout function of TX also affects the XATMI interface:

tx_set_transaction_timeout( )
A function that specifies the time interval in which the transaction must complete.

There are two types of timeout when using XATMI and TX: one is associated with the duration of a transaction from start to finish; the other is associated with the maximum length of time a blocking call remains blocked before the caller regains control. The first kind of timeout is specified when a transaction is started with the TX API's *tx_begin*( ) (see the **TX** (Transaction Demarcation) specification for details). The second kind of timeout can occur when using an XATMI communication routine (for example *tpcall*( ), *tpconnect*( ) or *tprecv*( )). Callers of these routines typically block when awaiting data that has yet to arrive, although they can also block trying to send data (for example, if transmission buffers are full). When the caller is not part of any global (TX) transaction, the maximum amount of time a caller remains blocked is determined in an XATMI provider-specific manner. Routines that return control after either type of timeout has occurred return a particular error code that signifies a timeout event.

Of the two timeout mechanisms, blocking timeouts are performed by default when the caller is not in transaction mode. When a client or server is in transaction mode, it is subject to the timeout value with which the transaction was started and is not subject to any blocking timeout value specified by the XATMI provider.

When a timeout occurs, replies to asynchronous requests may be dropped. That is, if a process is waiting for a particular asynchronous reply and a transaction timeout occurs, the descriptor for that reply becomes invalid and that reply is silently discarded. Similarly, if a transaction timeout occurs during a conversation with a service, an event is generated on the associated connection descriptor, that descriptor becomes invalid, and data may be lost. On the other hand, if a blocking timeout occurs, both types of descriptor remain valid and the waiting process can re-issue the call to await the reply.

3.7.2 Effect on Service Calls

Services are either invoked in a global transaction or outside a global transaction. If a requester invokes a service as part of its transaction, the service can participate in only that transaction and the service does not call any transaction demarcation functions. If the requester invokes a service outside a transaction, the service routine can originate and complete any number of transactions using the TX (Transaction Demarcation) interface.

In order for a transaction propagated to a service routine to successfully commit, the service routine must first receive all outstanding replies for requests that it generated as well as close any outgoing connections to conversational services that it opened.

3.8 Naming Rules

The XATMI interface uses three kinds of names: *service names*, *buffer type names*, and *buffer sub-type names*. Names are passed in the interface as null-terminated character strings. Three buffer type names are defined in this specification; other names are application-defined.

Names that meet the following rules are guaranteed to be portable and interoperable across implementations that conform to the XATMI interface.

- A name is composed of one or more characters from the set of letters (A-Z, a-z), digits (0-9), and underscore (_).
- A name must begin with a letter or underscore.
- The case of letters in a name is significant.
- The first 15 characters determine the service name.
- A buffer type name can contain up to 8 characters.
- A buffer sub-type name can contain up to 16 characters.
- A name is terminated by a null (0x00) character or by the first space encountered, or by reaching the length limit for the kind of name.

Chapter 4

The <xatmi.h> Header

This chapter specifies C-language structure element definitions, argument values, and return codes to which conforming products must adhere. These, plus the function prototypes for the interface routines defined in the next chapter, are the minimum required contents of the C-language header file **<xatmi.h>**.

4.1 Flag Bits

The following constants are the flag bits defined for the C-language XATMI routines:

```
#define TPNOBLOCK  0x00000001
#define TPSIGRSTRT 0x00000002
#define TPNOREPLY  0x00000004
#define TPNOTRAN   0x00000008
#define TPTRAN     0x00000010
#define TPNOTIME   0x00000020
#define TPGETANY   0x00000080
#define TPNOCHANGE 0x00000100
#define TPCONV     0x00000400
#define TPSENDONLY 0x00000800
#define TPRECVONLY 0x00001000
```

4.2 Service Return Value

The following constants are the names defined for the *rval* parameter to *tpreturn*( ):

```
#define TPFAIL    0x0001
#define TPSUCCESS 0x0002
```

4.3 Service Information Structure

The following elements are members of the **TPSVCINFO** structure passed into a service routine when it is dispatched:

```
#define XATMI_SERVICE_NAME_LENGTH  x      /* where x must be >= 15 */
   char name[XATMI_SERVICE_NAME_LENGTH];
   char *data;
   long len;
   long flags;
   int  cd;
```

4.4 Global Variables

The following definitions are the global variables used by the C language XATMI routines:

```
extern int tperrno;
extern long tpurcode;
```

4.5 Error Values

The following constants are the names defined for the *tperrno* global variable:

```
#define TPEBADDESC  2
#define TPEBLOCK    3
#define TPEINVAL    4
#define TPELIMIT    5
#define TPENOENT    6
#define TPEOS       7
#define TPEPROTO    9
#define TPESVCERR  10
#define TPESVCFAIL 11
#define TPESYSTEM  12
#define TPETIME    13
#define TPETRAN    14
#define TPGOTSIG   15
#define TPEITYPE   17
#define TPEOTYPE   18
#define TPEEVENT   22
#define TPEMATCH   23
```

4.6 XATMI Events

The following constants are the names defined for events returned during conversational communication:

```
#define TPEV_DISCONIMM 0x0001
#define TPEV_SVCERR    0x0002
#define TPEV_SVCFAIL   0x0004
#define TPEV_SVCSUCC   0x0008
#define TPEV_SENDONLY  0x0020
```

4.7 Typed Buffer Constants

The following constants are the names of the X/Open defined typed buffers:

```
#define X_OCTET   "X_OCTET"
#define X_C_TYPE  "X_C_TYPE"
#define X_COMMON  "X_COMMON"
```

Chapter 5

C Reference Manual Pages

This chapter contains the C-language reference manual pages for the XATMI communication API for transaction processing. The reference manual pages appear, in alphabetical order, for each C-language function in the XATMI interface.

The symbolic constants and error names are described in the **<xatmi.h>** header (see Chapter 4).

NAME

tpacall — send a service request

SYNOPSIS

```
#include <xatmi.h>

int tpacall(char *svc, char *data, long len, long flags)
```

DESCRIPTION

The function *tpacall*( ) sends a request message to the service named by *svc*. If *data* is non-NULL, it must point to a buffer previously allocated by *tpalloc*( ) and *len* should specify the amount of data in the buffer that should be sent. Note that if *data* points to a buffer of a type that does not require a length to be specified, *len* is ignored (and may be 0). If *data* points to a buffer that does require a length, *len* must not be zero. If *data* is NULL, *len* is ignored and a request is sent with no data portion. The type and sub-type of *data* must match one of the types and sub-types recognised by *svc*. Note that for each request sent while in transaction mode, a corresponding reply must ultimately be received.

The valid *flags* are as follows:

TPNOTRAN
: If the caller is in transaction mode and this flag is set, when *svc* is invoked, it is not performed on behalf of the caller's transaction. If *svc* does not support transactions, this flag must be set when the caller is in transaction mode. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other). If a service fails that was invoked with this flag, the caller's transaction is not affected.

TPNOREPLY
: This setting informs *tpacall*( ) that a reply is not expected. When TPNOREPLY is set, the function returns 0 on success, where 0 is an invalid descriptor. When the caller is in transaction mode, this setting cannot be used unless TPNOTRAN is also set.

TPNOBLOCK
: The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). When TPNOBLOCK is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME
: This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is reissued.

RETURN VALUE

Upon successful completion, *tpacall*( ) returns a descriptor that can be used to receive the reply of the request sent. Otherwise it returns –1 and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpacall*( ) fails and sets *tperrno* to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

[TPEINVAL]
: Invalid arguments were given (for example, *svc* is NULL, *data* does not point to space allocated with *tpalloc*( ) or *flags* are invalid).

[TPENOENT]
Cannot send to *svc* because it does not exist.

[TPEITYPE]
The type and sub-type of *data* are not of the allowed types and sub-types that *svc* accepts.

[TPELIMIT]
The caller's request was not sent because the maximum number of outstanding asynchronous requests has been reached.

[TPETRAN]
svc does not support transactions and TPNOTRAN was not set.

[TPETIME]
A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. If a transaction timeout occurred, any attempts to send new requests or receive outstanding replies fail with [TPETIME] until the transaction has been rolled back.

[TPEBLOCK]
A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]
tpacall( ) was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpalloc( ), *tpcall*( ), *tpcancel*( ), *tpgetrply*( ).

NAME

tpadvertise — advertise a service name

SYNOPSIS

```
#include <xatmi.h>

int tpadvertise(char *svcname, void (*func)(TPSVCINFO *))
```

DESCRIPTION

The function *tpadvertise*( ) allows a server to advertise the services that it offers. By default, a server's services are advertised when it is booted and unadvertised when it is shut down.

The function *tpadvertise*( ) advertises *svcname* for the server. The argument *svcname* should be 15 characters or fewer, but cannot be NULL or the NULL string (""). Longer names are accepted and truncated to 15 characters. Users should make sure that truncated names do not match other service names. The argument *func* is the address of a service function. This function is invoked whenever a request for *svcname* is received by the server. The argument *func* cannot be NULL.

If *svcname* is already advertised for the server and *func* matches its current function, *tpadvertise*( ) returns success (this includes truncated names that match already advertised names). However, if *svcname* is already advertised for the server but *func* does not match its current function, an error is returned (this can happen if truncated names match already advertised names).

RETURN VALUE

The function *tpadvertise*( ) returns –1 on error and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpadvertise*( ) fails and sets *tperrno* to one of the following values:

[TPEINVAL]
: The argument *svcname* is NULL or the NULL string (""), or *func* is NULL.

[TPELIMIT]
: The argument *svcname* cannot be advertised because of space limitations.

[TPEMATCH]
: The argument *svcname* is already advertised for the server but with a function other than *func*. Although the function fails, *svcname* remains advertised with its current function (that is, *func* does not replace the current function).

[TPEPROTO]
: The function *tpadvertise*( ) was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpservice( ), *tpunadvertise*( ).

NAME

tpalloc — allocate a typed buffer

SYNOPSIS

```
#include <xatmi.h>

char * tpalloc(char *type, char *subtype, long size)
```

DESCRIPTION

The function *tpalloc*( ) returns a pointer to a buffer of type *type*. Depending on the type of buffer, both *subtype* and *size* are optional.

If multiple subtypes are available for a particular buffer type, *subtype* must be specified when *tpalloc*( ) is called. If the type specified does not have a subtype, **subtype* is ignored (and may be null). The allocated buffer is at least as large as *size*.

Note that only the first eight bytes of *type* and the first 16 bytes of *subtype* are significant.

Because some buffer types require initialisation before they can be used, *tpalloc*( ) initialises a buffer (in a communication-resource-manager-specific manner) after it is allocated and before it is returned. Thus, the buffer returned to the caller is ready for use. Note that unless the initialisation processing cleared the buffer, the buffer is not initialised to zeros by *tpalloc*( ).

RETURN VALUE

Upon successful completion, *tpalloc*( ) returns a pointer to a buffer of the appropriate type aligned on a **long** word. Otherwise it returns NULL and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpalloc*( ) fails and sets *tperrno* to one of the following values:

[TPEINVAL]
: Invalid arguments were given (for example, *type* is NULL).

[TPENOENT]
: Unknown *type* and/or *subtype*.

[TPEPROTO]
: *tpalloc*( ) was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

APPLICATION USAGE

If buffer initialisation processing fails, the allocated buffer is freed and *tpalloc*( ) fails returning NULL.

This function should not be used in concert with *malloc*( ), *realloc*( ) or *free*( ) in the C library (for example, a buffer allocated with *tpalloc*( ) should not be freed with *free*( )).

SEE ALSO

tpfree( ), *tprealloc*( ), *tptypes*( ).

NAME

tpcall — send a service request and synchronously await its reply

SYNOPSIS

```
#include <xatmi.h>

int tpcall(char *svc, char *idata, long ilen,
          char **odata, long *olen, long flags)
```

DESCRIPTION

The function *tpcall*( ) sends a request and synchronously awaits its reply. A call to this function is the same as calling *tpacall*( ) immediately followed by *tpgetrply*( ). The function *tpcall*( ) sends a request to the service named by *svc*. The data portion of a request is pointed to by *idata*, a buffer previously allocated by *tpalloc*( ). The argument *ilen* specifies how much of *idata* to send. Note that if *idata* points to a buffer of a type that does not require a length to be specified, *ilen* is ignored (and may be 0). If *data* points to a buffer that does require a length, *len* must not be zero. Also, *idata* may be NULL in which case *ilen* is ignored. The type and sub-type of *idata* must match one of the types and sub-types recognised by *svc*.

odata is the address of a pointer to the buffer where a reply is read into, and the length of that reply is returned in **olen*. **odata* must point to a buffer originally allocated by *tpalloc*( ). If the same buffer is to be used for both sending and receiving, *odata* should be set to the address of *idata*. To determine whether a reply buffer changed in size, compare its (total) size before *tpcall*( ) was issued with **olen*. If **olen* is larger, the buffer has grown; otherwise, the buffer has not changed size. Also, if *idata* and **odata* were equal when *tpcall*( ) was invoked, and **odata* is changed, *idata* no longer points to a valid address. Note that **odata* may change for reasons other than the buffer's size increased. If **olen* is 0 upon return, the reply has no data portion and neither **odata* nor the buffer it points to were modified. It is an error for **odata* or *olen* to be NULL.

The valid *flags* are as follows:

TPNOTRAN
: If the caller is in transaction mode and this flag is set, when *svc* is invoked, it is not performed on behalf of the caller's transaction. If *svc* does not support transactions, this flag must be set when the caller is in transaction mode. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other). If a service fails that was invoked with this flag, the caller's transaction is not affected.

TPNOCHANGE
: By default, if a buffer is received that differs in type from the buffer pointed to by **odata*, **odata*'s buffer type changes to the received buffer's type so long as the receiver recognises the incoming buffer type. When this flag is set, the type of the buffer pointed to by **odata* is not allowed to change. That is, the type and sub-type of the received buffer must match the type and sub-type of the buffer pointed to by **odata*.

TPNOBLOCK
: The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Note that this flag applies only to the send portion of *tpcall*( ); the function may block waiting for the reply. When TPNOBLOCK is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME
: This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is reissued.

RETURN VALUE

Upon successful return from *tpcall*( ) or upon return where *tperrno* is set to [TPESVCFAIL], the *tpurcode* global contains an application-defined value that was sent as part of *tpreturn*( ). Otherwise, it returns –1 and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpcall*( ) fails and sets *tperrno* to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

[TPEINVAL]
: Invalid arguments were given (for example, *svc* is NULL or *flags* are invalid).

[TPENOENT]
: Cannot send to *svc* because it does not exist.

[TPEITYPE]
: The type and sub-type of *idata* are not of the allowed types and sub-types that *svc* accepts.

[TPEOTYPE]
: Either the type and sub-type of the reply are not known to the caller; or, TPNOCHANGE was set in *flags* and the type and sub-type of **odata* do not match the type and sub-type of the reply sent by the service. Neither **odata*, its contents nor **olen* are changed. If the service request was made on behalf of the caller's current transaction, the transaction is marked rollback-only since the reply is discarded.

[TPETRAN]
: The argument *svc* does not support transactions and TPNOTRAN was not set.

[TPETIME]
: A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. In either case, neither **odata*, its contents nor **olen* are changed. If a transaction timeout occurred, any attempts to send new requests or receive outstanding replies fail with [TPETIME] until the transaction has been rolled back.

[TPESVCFAIL]
: The service routine sending the caller's reply called *tpreturn*( ) with TPFAIL. This is an application-level failure. The contents of the service's reply, if one was sent, are available in the buffer pointed to by **odata*. If the service request was made on behalf of the caller's current transaction, the transaction is marked rollback-only. Note that so long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Such attempts should be made with TPNOTRAN set if they are to have any lasting effect. Any work performed on behalf of the caller's transaction is rolled back upon transaction completion.

[TPESVCERR]
: An error was encountered either in invoking a service routine or during its completion in *tpreturn*( ) (for example, bad arguments were passed). No reply data is returned when this error occurs (that is, neither **odata*, its contents nor **olen* are changed). If the service request was made on behalf of the caller's transaction, the transaction is marked rollback-only. Note that so long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Such attempts should be made with

TPNOTRAN set if they are to have any lasting effect. Any work performed on behalf of the caller's transaction is rolled back upon transaction completion.

[TPEBLOCK]
A blocking condition was found on the send portion of *tpcall*( ) and TPNOBLOCK was specified.

[TPGOTSIG]
A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]
tpcall( ) was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpalloc( ), *tpacall*( ), *tpgetrply*( ), *tpreturn*( ).

NAME

tpcancel — cancel a call descriptor for an outstanding reply

SYNOPSIS

```
#include <xatmi.h>

int tpcancel(int cd)
```

DESCRIPTION

The function *tpcancel*( ) cancels a call descriptor, *cd*, returned by *tpacall*( ). It is an error to attempt to cancel a call descriptor associated with a global transaction.

Upon successful return, *cd* is no longer valid and any reply received (by the communication resource manager) on behalf of *cd* is silently discarded.

RETURN VALUE

tpcancel( ) returns −1 on error and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpcancel*( ) fails and sets *tperrno* to one of the following values:

[TPEBADDESC]
The argument *cd* is an invalid descriptor.

[TPETRAN]
The argument *cd* is associated with the caller's global transaction. *cd* remains valid and the caller's current transaction is not affected.

[TPEPROTO]
The function *tpcancel*( ) was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpacall( ).

NAME

tpconnect — establish a conversational service connection

SYNOPSIS

```
#include <xatmi.h>

int tpconnect(char *svc, char *data, long len, long flags)
```

DESCRIPTION

The function *tpconnect*( ) allows a program to set up a half-duplex connection to a conversational service, *svc*.

As part of setting up a connection, the caller can pass application-defined data to the receiving service routine. If the caller chooses to pass data, *data* must point to a buffer previously allocated by *tpalloc*( ). *len* specifies how much of the buffer to send. Note that if *data* points to a buffer of a type that does not require a length to be specified, *len* is ignored (and may be 0). If *data* points to a buffer that does require a length, *len* must not be zero. Also, *data* can be NULL in which case *len* is ignored (no application data is passed to the conversational service). The type and sub-type of *data* must match one of the types and sub-types recognised by *svc*. Because the conversational service receives *data* and *len* via the TPSVCINFO structure upon invocation, the service does not call *tprecv*( ) to get the data sent by *tpconnect*( ).

The valid *flags* are as follows:

TPNOTRAN
: If the caller is in transaction mode and this flag is set, when *svc* is invoked, it is not performed on behalf of the caller's transaction. If *svc* does not support transactions, this flag must be set when the caller is in transaction mode. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other). If a service fails that was invoked with this flag, the caller's transaction is not affected.

TPSENDONLY
: The caller wants the connection to be set up initially such that it can send data and the called service can only receive data (that is, the caller initially has control of the connection). Either TPSENDONLY or TPRECVONLY must be specified.

TPRECVONLY
: The caller wants the connection to be set up initially such that it can only receive data and the called service can send data (that is, the service being called initially has control of the connection). Either TPSENDONLY or TPRECVONLY must be specified.

TPNOBLOCK
: The connection is not established and the data is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). When TPNOBLOCK is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME
: This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is reissued.

RETURN VALUE

Upon successful completion, *tpconnect*( ) returns a descriptor that is used to refer to the connection in subsequent calls. Otherwise it returns –1 and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpconnect*( ) fails and sets *tperrno* to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

[TPEINVAL]
: Invalid arguments were given (for example, *svc* is NULL, *data* is non-NULL and does not point to a buffer allocated by *tpalloc*( ), TPSENDONLY or TPRECVONLY was not specified in *flags*, or *flags* are otherwise invalid).

[TPENOENT]
: Cannot initiate a connection to *svc* because it does not exist.

[TPEITYPE]
: The type and subtype of *data* are not of the allowed types and subtypes that *svc* accepts.

[TPELIMIT]
: The caller's request was not sent because the maximum number of outstanding connections has been reached.

[TPETRAN]
: The argument *svc* does not support transactions and TPNOTRAN was not set.

[TPETIME]
: A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. If a transaction timeout occurred, any attempts to send or receive messages on any connections or to start a new connection fail with [TPETIME] until the transaction has been rolled back.

[TPEBLOCK]
: A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
: A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]
: *tpconnect*( ) was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpalloc( ), *tpdiscon*( ), *tprecv*( ), *tpsend*( ), *tpservice*( ).

NAME

tpdiscon — terminate a conversational service connection abortively

SYNOPSIS

```
#include <xatmi.h>

int tpdiscon(int cd)
```

DESCRIPTION

The function *tpdiscon*( ) immediately terminates the connection specified by *cd* and generates a TPEV_DISCONIMM event on the other end of the connection.

The function *tpdiscon*( ) can be called only by the initiator of the conversation. *tpdiscon*( ) cannot be called within a conversational service on the descriptor with which it was invoked. Rather, a conversational service must use *tpreturn*( ) to signify that it has completed its part of the conversation. Similarly, even though a program communicating with a conversational service can issue *tpdiscon*( ), the preferred way is to let the service terminate the connection in *tpreturn*( ); doing so ensures correct results.

The function *tpdiscon*( ) causes the connection to be terminated immediately (that is, abortively rather than orderly). Any data that has not yet reached its destination may be lost. *tpdiscon*( ) can be issued even when the program on the other end of the connection is participating in the caller's transaction. In this case, the transaction must be rolled back. Also, the caller does not need to have control of the connection when *tpdiscon*( ) is called.

RETURN VALUE

The function *tpdiscon*( ) returns –1 on error and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpdiscon*( ) fails and sets *tperrno* to one of the following values:

[TPEBADDESC]
: The argument *cd* is invalid or is the descriptor with which a conversational service was invoked.

[TPETIME]
: A timeout occurred. The descriptor is no longer valid.

[TPEPROTO]
: The function *tpdiscon*( ) was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpconnect( ), *tprecv*( ), *tpreturn*( ), *tpsend*( ).

NAME

tpfree — free a typed buffer

SYNOPSIS

```
#include <xatmi.h>

void tpfree(char *ptr)
```

DESCRIPTION

The argument to *tpfree*( ) is a pointer to a buffer previously obtained by either *tpalloc*( ) or *tprealloc*( ). If *ptr* is NULL, no action occurs. Undefined results occur if *ptr* does not point to a typed buffer (or if it points to space previously freed with *tpfree*( )). Inside service routines, *tpfree*( ) returns and does not free the buffer if *ptr* points to the buffer passed into a service routine.

Some buffer types require state information or associated data to be removed as part of freeing a buffer. *tpfree*( ) removes any of these associations (in a communication-resource-manager-specific manner) before a buffer is freed.

Once *tpfree*( ) returns, *ptr* should not be passed as an argument to any XATMI routine or used in any other manner.

RETURN VALUE

The function *tpfree*( ) does not return any value to its caller. Therefore, it is declared as a **void**.

APPLICATION USAGE

This function should not be used in concert with *malloc*( ), *realloc*( ) or *free*( ) in the C library (for example, a buffer allocated with *tpalloc*( ) should not be freed with *free*( )).

SEE ALSO

tpalloc( ), *tprealloc*( ).

NAME

tpgetrply — get a reply from a previous service request

SYNOPSIS

```
#include <xatmi.h>

int tpgetrply(int *cd, char **data, long *len, long flags)
```

DESCRIPTION

The function *tpgetrply*( ) returns a reply from a previously-sent service request. This function's first argument, *cd*, points to a call descriptor returned by *tpacall*( ). By default, the function waits until the reply matching **cd* arrives or a timeout occurs.

data must be the address of a pointer to a buffer previously allocated by *tpalloc*( ) and *len* should point to a **long** that *tpgetrply*( ) sets to the amount of data successfully received. *tpgetrply*( ) ensures that the request fits into the specified buffer by growing the buffer if necessary. Upon successful return, **data* points to a buffer containing the reply and **len* contains the size of the data. Note that **data* may have changed upon return for reasons other than an increase in the size of the buffer. If **len* is greater than the total size of the buffer before the call, the buffer's new size is **len*. If **len* is 0, then the reply dequeued has no data portion and neither **data* nor the buffer it points to were modified. It is an error for **data* or *len* to be NULL.

The valid *flags* are as follows:

TPGETANY
: This flag signifies that *tpgetrply*( ) should ignore the descriptor pointed to by *cd*, return any reply available and set *cd* to point to the call descriptor for the reply returned. If no replies exist, by default *tpgetrply*( ) waits for one to arrive.

TPNOCHANGE
: By default, if a buffer is received that differs in type from the buffer pointed to by **data*, then **data*'s buffer type changes to the received buffer's type so long as the receiver recognises the incoming buffer type. When this flag is set, the type of the buffer pointed to by **data* is not allowed to change. That is, the type and sub-type of the received buffer must match the type and sub-type of the buffer pointed to by **data*.

TPNOBLOCK
: *tpgetrply*( ) does not wait for the reply to arrive. If the reply is available, *tpgetrply*( ) gets the reply and returns. When this flag is not specified and a reply is not available, the caller blocks until the reply arrives or a timeout occurs (either transaction or blocking timeout).

TPNOTIME
: This flag signifies that the caller is willing to block indefinitely for its reply and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is reissued.

Except as noted below, **cd* is no longer valid after its reply is received.

RETURN VALUE

Upon successful return from *tpgetrply*( ) or upon return where *tperrno* is set to [TPESVCFAIL], the *tpurcode* global contains an application-defined value that was sent as part of *tpreturn*( ). Otherwise, it returns –1 and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpgetrply*( ) fails and sets *tperrno* as indicated below. Note that if TPGETANY is not set, **cd* is invalidated unless otherwise stated. If TPGETANY is set, *cd* points to the descriptor for the reply on which the failure occurred; if an error occurred before a reply could be retrieved, *cd* points to 0, unless otherwise stated. Also, the failure does not affect the caller's transaction, if one exists, unless otherwise stated.

[TPEINVAL]
: Invalid arguments were given (for example, *cd*, *data*, **data* or *len* is NULL or *flags* are invalid). If *cd* is non-NULL, it is still valid after this error and the reply remains outstanding.

[TPEBADDESC]
: The argument *cd* points to an invalid descriptor.

[TPEOTYPE]
: Either the type and sub-type of the reply are not known to the caller, or TPNOCHANGE was set in *flags* and the type and sub-type of **data* do not match the type and sub-type of the reply sent by the service. In either case, neither **data*, its contents nor **len* are changed. If the reply was to be received on behalf of the caller's current transaction, the transaction is marked rollback-only since the reply is discarded.

[TPETIME]
: A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. In either case, neither **data*, its contents nor **len* are changed. The argument **cd* remains valid unless the caller is in transaction mode (and TPGETANY was not set). If a transaction timeout occurred, any attempts to send new requests or receive outstanding replies fail with [TPETIME] until the transaction has been rolled back.

[TPESVCFAIL]
: The service routine sending the caller's reply called *tpreturn*( ) with TPFAIL. This is an application-level failure. The contents of the service's reply, if one was sent, are available in the buffer pointed to by **data*. If the reply was received on behalf of the caller's transaction, the transaction is marked rollback-only. Note that so long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Such attempts should be made with TPNOTRAN set if they are to have any lasting effect. Any work performed on behalf of the caller's transaction is rolled back upon transaction completion.

[TPESVCERR]
: An error was encountered either in invoking a service routine or during its completion in *tpreturn*( ) (for example, bad arguments were passed). No reply data is returned when this error occurs (that is, neither **data*, its contents nor **len* are changed). If the reply was received on behalf of the caller's transaction, the transaction is marked rollback-only. Note that so long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Such attempts should be made with TPNOTRAN set if they are to have any lasting effect. Any work performed on behalf of the caller's transaction is rolled back upon transaction completion.

[TPEBLOCK]
: A blocking condition exists and TPNOBLOCK was specified. The argument **cd* remains valid.

[TPGOTSIG]
A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]
tpgetrply( ) was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpacall( ), *tpalloc*( ), *tpreturn*( ).

NAME

tprealloc — change the size of a typed buffer

SYNOPSIS

```
#include <xatmi.h>

char * tprealloc(char *ptr, long size)
```

DESCRIPTION

The function *tprealloc*( ) changes the size of the buffer pointed to by *ptr* to *size* bytes and returns a pointer to the new (possibly moved) buffer. As with *tpalloc*( ), the size of the buffer is at least as large as *size*. A buffer's type remains the same after it is reallocated. After this function returns successfully, the returned pointer should be used to reference the buffer; *ptr* should no longer be used. The buffer's contents do not change up to the lesser of the new and old sizes.

Some buffer types require initialisation before they can be used. *tprealloc*( ) reinitialises a buffer (in a communication-resource-manager-specific manner) after it is reallocated and before it is returned. Thus, the buffer returned to the caller is ready for use.

RETURN VALUE

Upon successful completion, *tprealloc*( ) returns a pointer to a buffer of the appropriate type aligned on a **long** word. Otherwise it returns NULL and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tprealloc*( ) fails and sets *tperrno* to one of the following values:

[TPEINVAL]
Invalid arguments were given (for example, *ptr* does not point to a buffer originally allocated by *tpalloc*( )).

[TPEPROTO]
tprealloc( ) was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

APPLICATION USAGE

If buffer reinitialisation fails, *tprealloc*( ) fails returning NULL and the contents of the buffer pointed to by *ptr* may not be valid.

This function should not be used in concert with *malloc*( ), *realloc*( ) or *free*( ) in the C library (for example, a buffer allocated with *tprealloc*( ) should not be freed with *free*( )).

SEE ALSO

tpalloc( ), *tpfree*( ), *tptypes*( ).

NAME

tprecv — receive a message in a conversational connection

SYNOPSIS

```
#include <xatmi.h>

int tprecv(int cd, char **data, long *len, long flags, long *revent)
```

DESCRIPTION

The function *tprecv*( ) is used to receive data sent across an open connection from another program. This function's first argument, *cd*, specifies on which open connection to receive data. *cd* is a descriptor returned from either *tpconnect*( ) or the TPSVCINFO parameter to the service. The second argument, *data*, is the address of a pointer to a buffer previously allocated by *tpalloc*( ).

Upon successful return, and for several event types, **data* points to the data received and **len* contains the size of the buffer. Note that if **len* is greater than the total size of the buffer before the call, the buffer's new size is **len*. If **len* is 0, no data was received and neither **data* nor the buffer it points to were modified. It is an error for *data*, **data* or *len* to be NULL.

tprecv( ) can be issued only by the program that does not have control of the connection.

The valid *flags* are as follows:

TPNOCHANGE
: By default, if a buffer is received that differs in type from the buffer pointed to by **data*, then **data*'s buffer type changes to the received buffer's type so long as the receiver recognises the incoming buffer type. When this flag is set, the type of the buffer pointed to by **data* is not allowed to change. That is, the type and sub-type of the received buffer must match the type and sub-type of the buffer pointed to by **data*.

TPNOBLOCK
: The function *tprecv*( ) does not wait for data to arrive. If data is already available to receive, *tprecv*( ) gets the data and returns. When this flag is not specified and data is not available to receive, the caller blocks until data arrives.

TPNOTIME
: This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is reissued.

If an event exists for the descriptor, *cd*, and *tprecv*( ) encounters no errors, the event type is returned in *revent*. Data can be received along with the TPEV_SVCSUCC, TPEV_SVCFAIL, and TPEV_SENDONLY events. Valid events for *tprecv*( ) are as follows:

TPEV_DISCONIMM
: Received by the subordinate of a conversation, this event indicates that the originator of the conversation has either issued an immediate disconnect on the connection by means of *tpdiscon*( ), or it issued *tpreturn*( ), *tx_commit*( ) or *tx_rollback*( ) with the connection still open. This event is also returned to the originator or subordinate when a connection is broken due to a communication error (for example, a server, machine, or network failure). Because this is an immediate disconnection notification (that is, abortive rather than orderly), data in transit may be lost. If the two programs were participating in the same transaction, the transaction is marked rollback-only. The descriptor used for the connection is no longer valid.

TPEV_SENDONLY

The program at the other end of the connection has relinquished control of the connection. The recipient of this event is allowed to send data but cannot receive any data until it relinquishes control.

TPEV_SVCERR

Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued *tpreturn*( ). *tpreturn*( ) encountered an error that precluded the service from returning successfully. For example, bad arguments may have been passed to *tpreturn*( ) or it may have been called while the service had open connections to other subordinates. Due to the nature of this event, any application-defined data or return code are not available. The connection has been terminated and *cd* is no longer a valid descriptor. If this event occurred as part of the recipient's transaction, the transaction is marked rollback-only.

TPEV_SVCFAIL

Received by the originator of a conversation, this event indicates that the subordinate service on the other end of the conversation has finished unsuccessfully as defined by the application (that is, it called *tpreturn*( ) with TPFAIL). If the subordinate service was in control of this connection when *tpreturn*( ) was called, it can pass a typed buffer back to the originator of the connection. As part of ending the service routine, the server has terminated the connection. Thus, *cd* is no longer a valid descriptor. If this event occurred as part of the recipient's transaction, the transaction is marked rollback-only.

TPEV_SVCSUCC

Received by the originator of a conversation, this event indicates that the subordinate service on the other end of the conversation has finished successfully as defined by the application (that is, it called *tpreturn*( ) with TPSUCCESS). As part of ending the service routine, the server has terminated the connection. Thus, *cd* is no longer a valid descriptor. If the recipient is in transaction mode, it can either commit (if it is also the initiator) or roll back the transaction causing the work done by the server (if also in transaction mode) to either commit or roll back.

RETURN VALUE

Upon return from *tprecv*( ) where *revent* is set to either TPEV_SVCSUCC or TPEV_SVCFAIL, the *tpurcode* global contains an application-defined value that was sent as part of *tpreturn*( ). The function *tprecv*( ) returns –1 on error and sets *tperrno* to indicate the error condition. Also, if an event exists and no errors were encountered, *tprecv*( ) returns –1 and *tperrno* is set to [TPEEVENT].

ERRORS

Under the following conditions, *tprecv*( ) fails and sets *tperrno* to one of the following values:

[TPEINVAL]

Invalid arguments were given (for example, *data* is not the address of a pointer to a buffer allocated by *tpalloc*( ) or *flags* are invalid).

[TPEBADDESC]

The argument *cd* is invalid.

[TPEOTYPE]

Either the type and sub-type of the incoming buffer are not known to the caller, or TPNOCHANGE was set in *flags* and the type and sub-type of **data* do not match the type and sub-type of the incoming buffer. In either case, neither **data*, its contents nor **len* are changed. If the conversation is part of the caller's current transaction, the transaction is marked rollback-only since the incoming buffer is discarded. When this error occurs, any event for *cd* is dropped and the conversation may now be in an indeterminate state. The caller should terminate the conversation.

[TPETIME]
A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. In either case, neither **data* nor its contents are changed. If a transaction timeout occurred, any attempts to send or receive messages on any connections or to start a new connection fail with [TPETIME] until the transaction has been rolled back.

[TPEEVENT]
An event occurred and its type is available in *revent*.

[TPEBLOCK]
A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]
tprecv( ) was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpalloc( ), *tpconnect*( ), *tpdiscon*( ), *tpsend*( ).

NAME

tpreturn — return from a service routine

SYNOPSIS

```
#include <xatmi.h>

void tpreturn(int rval, long rcode, char *data, long len, long flags)
```

DESCRIPTION

The function *tpreturn*( ) indicates that a service routine has completed. *tpreturn*( ) acts like a **return** statement in the C-language (that is, when *tpreturn*( ) is called, the service routine returns to the communication resource manager). It is recommended that *tpreturn*( ) be called from within the service routine dispatched by the communication resource manager to ensure correct return of control to the communication resource manager.

The function *tpreturn*( ) is used to send a service's reply message. If the program receiving the reply is waiting in either *tpcall*( ), *tpgetrply*( ), or *tprecv*( ), then after a successful call to *tpreturn*( ), the reply is available in the receiver's buffer.

For conversational services, *tpreturn*( ) also terminates the connection. That is, the service routine cannot call *tpdiscon*( ) directly. To ensure correct results, the program that connected to the conversational service should not call *tpdiscon*( ); rather, it should wait for notification that the conversational service has completed (that is, it should wait for one of the events, like TPEV_SVCSUCC or TPEV_SVCFAIL, sent by *tpreturn*( )).

If the service routine was in transaction mode, *tpreturn*( ) places the service's portion of the transaction in a state where it may be either committed or rolled back when the transaction is completed. A service may be invoked multiple times as part of the same transaction so it is not necessarily fully committed nor rolled back until either *tx_commit*( ) or *tx_rollback*( ) is called by the originator of the transaction.

The function *tpreturn*( ) should be called after receiving all replies expected from service requests initiated by the service routine. Otherwise, depending on the nature of the service, either a [TPESVCERR] error or a TPEV_SVCERR event is returned to the program that initiated communication with the service routine. Any outstanding replies that are not received are automatically dropped by the communication resource manager. In addition, the descriptors for those replies become invalid.

The function *tpreturn*( ) should be called after closing all connections initiated by the service. Otherwise, depending on the nature of the service, either a [TPESVCERR] or a TPEV_SVCERR event is returned to the program that initiated communication with the service routine. Also, an immediate disconnect event (that is, TPEV_DISCONIMM) is sent over all open connections to subordinates.

Concerning control of the connection, if the service routine does not have control over the connection with which it was invoked when it issues *tpreturn*( ), two outcomes are possible. Firstly, if the service routine calls *tpreturn*( ) with *rval* set to TPFAIL and *data* is NULL, then a TPEV_SVCFAIL event is sent to the originator of this conversation. Secondly, if any other invocation of *tpreturn*( ) is used, a TPEV_SVCERR event is sent to the originator.

Since a conversational service has only one open connection that it did not initiate, the communication resource manager knows over which descriptor data (and any event) should be sent. For this reason, a descriptor is not passed to *tpreturn*( ).

The argument *rval* can be set to one of the following:

TPSUCCESS
: The service has terminated successfully. If data is present, it is sent (barring any failures processing the return). If the caller is in transaction mode, *tpreturn*( ) places the caller's portion of the transaction in a state such that it can be committed when the transaction ultimately commits. Note that a call to *tpreturn*( ) does not necessarily finalise an entire transaction. Also, even though the caller indicates success, if there are any outstanding replies or open connections, or if any work done within the service caused its transaction to be marked rollback-only, then a failed message is sent (that is, the recipient of the reply receives a [TPESVCERR] indication or a TPEV_SVCERR event). Note that if a transaction becomes rollback-only while in the service routine for any reason, *rval* should be set to TPFAIL. If TPSUCCESS is specified for a conversational service, a TPEV_SVCSUCC event is generated.

TPFAIL
: The service has terminated unsuccessfully from an application standpoint. An error is reported to the program receiving the reply. That is, the call to get the reply fails and the recipient receives a [TPSVCFAIL] indication or a TPEV_SVCFAIL event. If the caller is in transaction mode, *tpreturn*( ) marks the transaction as rollback-only (note that the transaction may already be marked rollback-only). Barring any failures in processing the return, the caller's data is sent, if present. One reason for not sending the caller's data is when a transaction timeout has occurred. In this case, the program waiting for the reply receives an error of [TPETIME].

If *rval* does not contain one of these two values, TPFAIL is assumed.

An application-defined return code, *rcode*, may be sent to the program receiving the service reply. This code is sent regardless of the setting of *rval* as long as a reply can be successfully sent (that is, as long as the receiving call returns success or [TPESVCFAIL], or receives one of the events TPEV_SVCSUCC or TPEV_SVCFAIL). The value of *rcode* is available to the receiver in the variable *tpurcode*.

data points to the data portion of a reply to be sent. If *data* is non-NULL, it must point to a buffer previously obtained by a call to *tpalloc*( ). If this is the same buffer passed to the service routine upon its invocation, its disposition is up to the communication resource manager; the service routine writer does not have to worry about whether it is freed or not. In fact, any attempt by the user to free this buffer fails. However, if the buffer passed to *tpreturn*( ) is not the same one with which the service is invoked, *tpreturn*( ) frees that buffer. *len* specifies the amount of the data buffer to be sent. If *data* points to a buffer that does not require a length to be specified, *len* is ignored (and may be 0). If *data* points to a buffer that does require a length, *len* must not be zero.

If *data* is NULL, *len* is ignored. In this case, if a reply is expected by the program that invoked the service, a reply is sent with no data portion. If no reply is expected, *tpreturn*( ) frees *data* as necessary and returns sending no reply.

Currently, *flags* are reserved for future use and must be set to 0.

If the service is conversational, there are two cases where the data portion is not transmitted:

- If the connection has already been terminated when the call is made (that is, the caller has received TPEV_DISCONIMM on the connection), this call simply ends the service routine and rolls back the current transaction, if one exists. In this case, the caller's data cannot be transmitted.

- If the caller does not have control of the connection, either TPEV_SVCFAIL or TPEV_SVCERR is sent to the originator of the connection as described above. Regardless of which event the originator receives, no data is transmitted; however, if the originator receives the TPEV_SVCFAIL event, the return code is available in the originator's *tpurcode* variable.

RETURN VALUE

A service routine does not return any value to its caller, the communication resource manager dispatcher; thus, it is declared as a **void**. Service routines, however, are expected to terminate using *tpreturn*( ). If a service routine returns without using *tpreturn*( ) (that is, it uses the C-language **return** statement or ''falls out of the function''), the server returns a service error to the service requester. In addition, all open connections to subordinates are disconnected immediately, and any outstanding asynchronous replies are dropped. If the server was in transaction mode at the time of failure, the transaction is marked rollback-only. Note also that if *tpreturn*( ) is used outside a service routine (for example, by routines that are not services), it returns having no effect.

ERRORS

Since *tpreturn*( ) ends the service routine, any errors encountered either in handling arguments or in processing cannot be indicated to the function's caller. Such errors cause *tperrno* to be set to [TPESVCERR] for a program receiving the service's outcome via either *tpcall*( ) or *tpgetrply*( ), and cause the event, TPEV_SVCERR, to be sent over the conversation to a program using *tpsend*( ) or *tprecv*( ).

SEE ALSO

tpalloc( ), *tpcall*( ), *tpconnect*( ), *tpdiscon*( ), *tpgetrply*( ), *tprecv*( ), *tpsend*( ), *tpservice*( ).

NAME

tpsend — send a message in a conversational connection

SYNOPSIS

```
#include <xatmi.h>

int tpsend(int cd, char *data, long len, long flags, long *revent)
```

DESCRIPTION

The function *tpsend*( ) is used to send data across an open connection to another program. The caller must have control of the connection. This function's first argument, *cd*, specifies the open connection over which data is sent. *cd* is a descriptor returned from either *tpconnect*( ) or the TPSVCINFO parameter passed to a conversational service.

The second argument, *data*, must point to a buffer previously allocated by *tpalloc*( ). *len* specifies how much of the buffer to send. Note that if *data* points to a buffer of a type that does not require a length to be specified, *len* is ignored (and may be 0). If *data* points to a buffer that does require a length, *len* must not be zero. Also, *data* can be NULL in which case *len* is ignored (no application data is sent — this might be done, for instance, to grant control of the connection without transmitting any data). The type and sub-type of *data* must match one of the types and sub-types recognised by the other end of the connection.

The valid *flags* are as follows:

TPRECVONLY
: This flag signifies that, after the caller's data is sent, the caller gives up control of the connection (that is, the caller cannot issue any more *tpsend*( ) calls). When the receiver at the other end of the connection receives the data sent by *tpsend*( ), it also receives an event (TPEV_SENDONLY) indicating that it has control of the connection (and cannot issue more any *tprecv*( ) calls).

TPNOBLOCK
: The data and any events are not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). When TPNOBLOCK is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME
: This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is reissued.

If an event exists for the descriptor, *cd*, *tpsend*( ) fails without sending the caller's data. The event type is returned in *revent*. Valid events for *tpsend*( ) are as follows:

TPEV_DISCONIMM
: Received by the subordinate of a conversation, this event indicates that the originator of the conversation has either issued an immediate disconnect on the connection via *tpdiscon*( ), or it issued *tpreturn*( ), *tx_commit*( ) or *tx_rollback*( ) with the connection still open. This event is also returned to the originator or subordinate when a connection is broken due to a communication error (for example, a server, machine, or network failure).

TPEV_SVCERR
: Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued *tpreturn*( ) without having control of the conversation. In addition, *tpreturn*( ) was issued in a manner different from that described for TPEV_SVCFAIL below.

TPEV_SVCFAIL

Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued *tpreturn*( ) without having control of the conversation. In addition, *tpreturn*( ) was issued with the TPFAIL and no data (that is, *rval* was set to TPFAIL and *data* was NULL).

Because each of these events indicates an immediate disconnection notification (that is, abortive rather than orderly), data in transit may be lost. The descriptor used for the connection is no longer valid. If the two programs were participating in the same transaction, the transaction has been marked rollback-only.

RETURN VALUE

The function *tpsend*( ) returns –1 on error and sets *tperrno* to indicate the error condition. Upon return from *tpsend*( ) where *revent* is set to TPEV_SVCFAIL, the *tpurcode* global contains an application-defined value that was set as part of *tpreturn*( ).

ERRORS

Under the following conditions, *tpsend*( ) fails and sets *tperrno* to one of the following values:

[TPEINVAL]

Invalid arguments were given (for example, *data* does not point to a buffer allocated by *tpalloc*( ) or *flags* are invalid).

[TPEBADDESC]

The argument *cd* is invalid.

[TPETIME]

A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. In either case, neither **data*, its contents nor **len* are changed. If a transaction timeout occurred, any attempts to send or receive messages on any connections or to start a new connection fail with [TPETIME] until the transaction has been rolled back.

[TPEEVENT]

An event occurred. *data* is not sent when this error occurs. The event type is returned in *revent*.

[TPEBLOCK]

A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]

A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]

tpsend( ) was called in an improper context.

[TPESYSTEM]

A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]

An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpalloc( ), *tpconnect*( ), *tpdiscon*( ), *tprecv*( ), *tpreturn*( ).

NAME

tpservice — template for service routines

SYNOPSIS

```
#include <xatmi.h>

void tpservice(TPSVCINFO *svcinfo)
```

DESCRIPTION

The function *tpservice*( ) is the template for writing service routines. This template is used for services that receive requests via *tpcall*( ) or *tpacall*( ) routines as well as by services that communicate via *tpconnect*( ), *tpsend*( ) and *tprecv*( ) routines.

Service routines processing requests made via either *tpcall*( ) or *tpacall*( ) receive, at most, one incoming message (in the *data* element of *svcinfo*) and send, at most, one reply (upon exiting the service routine with *tpreturn*( )).

Conversational services, on the other hand, are invoked by connection requests with, at most, one incoming message along with a means of referring to the open connection. When a conversational service routine is invoked, either the connecting program or the conversational service may send and receive data as defined by the application. The connection is half-duplex in nature meaning that one side controls the conversation (that is, it sends data) until it explicitly gives up control to the other side of the connection.

Concerning transactions, service routines can participate in, at most, one transaction if invoked in transaction mode. As far as the service routine writer is concerned, the transaction ends upon returning from the service routine. If the service routine is not invoked in transaction mode, the service routine may originate as many transactions as it wants using *tx_begin*( ), *tx_commit*( ) and *tx_rollback*( ). Note that *tpreturn*( ) is not used to complete a transaction. Thus, it is an error to call *tpreturn*( ) with an outstanding transaction that originated within the service routine.

Service routines are invoked with one argument: *svcinfo*, a pointer to a service information structure. This structure includes the following members:

```
char        name [XATMI_SERVICE_NAME_LENGTH];
char        *data;
long        len;
long        flags;
int         cd;
```

The element **name** is populated with the service name that the requester used to invoke the service.

The setting of **flags** upon entry to a service routine indicates attributes that the service routine may want to note. The possible values for **flags** are as follows:

TPCONV
: A connection request for a conversation has been accepted and the descriptor for the conversation is available in **cd**. If not set, this is a request/response service and **cd** is not valid.

TPTRAN
: The service routine is in transaction mode.

TPNOREPLY
: The caller is not expecting a reply. This option is not set if TPCONV is set.

TPSENDONLY
The service is invoked such that it can send data across the connection and the program on the other end of the connection can only receive data. This flag is mutually exclusive with TPRECVONLY and may be set only when TPCONV is also set.

TPRECVONLY
The service is invoked such that it can only receive data from the connection and the program on the other end of the connection can send data. This flag is mutually exclusive with TPSENDONLY and may be set only when TPCONV is also set.

The element **data** points to the data portion of a request message and **len** is the length of the data. The buffer pointed to by **data** was allocated by *tpalloc*( ) in the communication resource manager. This buffer may be grown by the user with *tprealloc*( ); however, it cannot be freed by the user. It is recommended that this buffer be the one passed to *tpreturn*( ) when the service ends. If a different buffer is passed to those routines, that buffer is freed by them. Note that the buffer pointed to by **data** is overwritten by the next service request even if this buffer is not passed to *tpreturn*( ). The element **data** may be NULL if no data accompanied the request. In this case, **len** is 0.

When TPCONV is set in **flags**, **cd** is the connection descriptor that can be used with *tpsend*( ) and *tprecv*( ) to communicate with the program that initiated the conversation.

RETURN VALUE

A service routine does not return any value to its caller, the communication resource manager dispatcher; thus, it is declared as a **void**. Service routines, however, are expected to terminate using *tpreturn*( ). If a service routine returns without using *tpreturn*( ) (that is, it uses the C-language **return** statement or ‘‘falls out of the function’’), the server returns a service error to the service requester. In addition, all open connections to subordinates are disconnected immediately, and any outstanding asynchronous replies are dropped. If the server was in transaction mode at the time of failure, the transaction is marked rollback-only. Note also that if *tpreturn*( ) is used outside a service routine (for example, by routines that are not services), it returns having no effect.

ERRORS

Since *tpreturn*( ) ends the service routine, any errors encountered either in handling arguments or in processing cannot be indicated to the function’s caller. Such errors cause *tperrno* to be set to [TPESVCERR] for a program receiving the service’s outcome via either *tpcall*( ) or *tpgetrply*( ), and cause the event, TPEV_SVCERR, to be sent over the conversation to a program using *tpsend*( ) or *tprecv*( ).

SEE ALSO

tpalloc( ), *tpcall*( ), *tpconnect*( ), *tpgetrply*( ), *tprecv*( ), *tpreturn*( ), *tpsend*( ).

NAME

tptypes — determine information about a typed buffer

SYNOPSIS

```
#include <xatmi.h>

long tptypes(char *ptr, char *type, char *subtype)
```

DESCRIPTION

The function *tptypes*( ) takes as its first argument a pointer to a data buffer and returns the type and subtype of that buffer in its second and third arguments, respectively. *ptr* must point to a buffer obtained from *tpalloc*( ). If *type* and *subtype* are non-NULL, the function populates the character arrays to which they point with the names of the buffer's type and subtype, respectively. If the names are of their maximum length (8 for *type*, 16 for *subtype*), the character array is not null-terminated. If no subtype exists, the array pointed to by *subtype* contains a NULL string ("").

Note that only the first eight bytes of *type* and the first 16 bytes of *subtype* are populated.

RETURN VALUE

Upon success, *tptypes*( ) returns the size of the buffer. Otherwise, it returns –1 upon failure and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tptypes*( ) fails and sets *tperrno* to one of the following values:

[TPEINVAL]
Invalid arguments were given (for example, *ptr* does not point to a typed buffer).

[TPEPROTO]
tptypes( ) was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpalloc( ), *tpfree*( ), *tprealloc*( ).

NAME

tpunadvertise — unadvertise a service name

SYNOPSIS

```
#include <xatmi.h>

int tpunadvertise(char *svcname)
```

DESCRIPTION

The function *tpunadvertise*( ) allows a server to unadvertise a service that it offers. By default, a server's services are advertised when it is booted and they are unadvertised when it is shutdown.

The function *tpunadvertise*( ) removes *svcname* as an advertised service for the server. The argument *svcname* cannot be NULL or the NULL string (""). Also, *svcname* should be 15 characters or fewer. Longer names are accepted and truncated to 15 characters. Care should be taken that truncated names do not match other service names.

RETURN VALUE

tpunadvertise( ) returns –1 on error and sets *tperrno* to indicate the error condition.

ERRORS

Under the following conditions, *tpunadvertise*( ) fails and sets *tperrno* to one of the following values:

[TPEINVAL]
svcname is NULL or the NULL string ("").

[TPENOENT]
svcname is not currently advertised by the server.

[TPEPROTO]
tpunadvertise( ) was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

tpadvertise( ).

Chapter 6

COBOL Language Interface Overview

The XATMI interface is the API to a CRM that supports a client-server paradigm in an X/Open DTP system. This interface offers the following programming models (see also the definitions in Chapter 2):

- The request/response service paradigm allows the writing of a structured service AP routine that receives a single request and may produce a single reply. The CRM automatically initialises the communication path to the server and automatically invokes the AP service routine. This paradigm simplifies the writing of the AP.
- The conversational service paradigm provides for the same automatic setup as for the request/response service paradigm, but lets the AP service routine exchange data with the requester multiple times and in an application-defined sequence. This is also a high-level paradigm; it simplifies the writing of the AP service routine but gives it more flexibility than a request/response service.

The CRM must know (typically from the local configuration) which paradigm is followed by the AP routine addressed by any given request for communication, because the RM must enforce a different state table in each case.

This chapter gives an overview of the COBOL interface; it describes each paradigm: its attributes, the XATMI routines available in each paradigm and their usage, and programming examples. This chapter also explains the COBOL API style and describes the concept of typed records. Chapter 7 contains reference manual pages for each routine in alphabetical order.

6.1 Index to Functions in the XATMI Interface

Name	Description	See
	Routines for Writing Service Routines	
TPSVCSTART	Start a service routine.	Section 6.4 on page 57.
TPRETURN	Return from a service routine.	Section 6.4 on page 57.
	Routines For Dynamically Advertising Service Names	
TPADVERTISE	Advertise a service name.	Section 6.5 on page 58.
TPUNADVERTISE	Unadvertise a service name.	Section 6.5 on page 58.
	Routines for request/response Services	
TPACALL	Send a service request.	Section 6.6 on page 59.
TPCALL	Send a service request and synchronously await its reply.	Section 6.6 on page 59.
TPCANCEL	Cancel a communication handle for an outstanding reply.	Section 6.6 on page 59.
TPGETRPLY	Get a reply from a previous service request.	Section 6.6 on page 59.
	Routines for Conversational Services	
TPCONNECT	Establish a conversational service connection.	Section 6.7 on page 60.
TPDISCON	Terminate a conversational service connection abortively.	Section 6.7 on page 60.
TPRECV	Receive a message in a conversational connection.	Section 6.7 on page 60.
TPSEND	Send a message in a conversational connection.	Section 6.7 on page 60.

Table 6-1 COBOL Language XATMI Functions

The TP* routines are the application interface provided by X/Open-compliant CRMs implementing the XATMI interface. An AP can call these routines.

An AP must call the TP* routines in accordance with Chapter 8. However, if an AP calls more than one CRM, or has more than one outstanding request or conversational connection using an XATMI CRM, its calls to each do not depend on the state of its dealings with any other RM, specific request, or connection.

6.2 COBOL API Style

Because COBOL has no type checking, the COBOL API for XATMI adopts a style that differs from the C API. It combines several individual parameters in a few records. By using the COPY statement to copy the description of those records in the program body, the risk of type mismatches is avoided (the COBOL compiler would not detect such mismatches). This is a generally accepted good practice in COBOL programming that is used to avoid problems resulting from type mismatches.

6.3 Typed Records

Before presenting an overview of the XATMI routines, the concept of typed records is first described. In order to send data to another AP, the sending AP first places the data in a *record*. The XATMI interface supports the notion of a *typed record*. A typed record is really a pair of COBOL records. The data record is defined in static storage and contains application data to be passed to another AP. An auxiliary type record accompanies the data record and it identifies to the CRM the interpretation and translation rules of the data record as it passes across heterogeneous machine boundaries. The auxiliary type record contains the data record's type, its optional subtype, and its optional length. Some record types require further specification via a subtype (for example, a particular record layout) and those of variable length require a length to be specified.

X/Open predefines two typed records for the COBOL XATMI interface that all implementations support (see Chapter 9). An AP can specify by the type and subtype that a record's structure is interpreted by the AP.

6.4 Service Paradigm

The *service paradigm* refers to the common aspect of automatic setup and invocation in both the request/response and conversational service paradigms.

Service routines are coded as COBOL language sub-programs. A service routine is invoked from implementation-specific dispatching code contained within a server. Handling of the communication path is independent of the service and is the responsibility of the CRM. From an application writer's viewpoint, communication between a requester and a service routine is utilised only for the duration of the routine invocation.

The TPSVCSTART routine is the first routine that a service should call upon invocation in order to retrieve information about the service request to be performed as well as any data sent by the requester. TPSVCSTART is used both for services that receive requests via TPCALL or TPACALL routines, and services that communicate via TPCONNECT, TPSEND and TPRECV routines.

TPRETURN is used to send a service's reply message. If an AP receiving the reply is waiting in either TPCALL, TPGETRPLY or TPRECV, then after a successful call to TPRETURN, the reply is available in the receiving AP's record.

Services can accept more than one kind of typed record. In fact, services can accept one record type on input and send a different record type in the response. The record types that a service accepts can be specified in the local configuration.

6.5 Service Names and Dynamic Advertising

The requester identifies a service with which it wishes to communicate in the service name parameter to TPACALL, TPCALL or TPCONNECT. This parameter is a character string (for example, "DEBIT" or "CREDIT") and is completely defined by the application.

When servers are started, they *advertise* the set of services that they offer (in an implementation-specific manner). At run time, service routines themselves can alter a server's set of service advertisements. AP service routines may choose to do this, for example, based upon time of day or information received as part of a service request.

TPADVERTISE allows a server to advertise a new service that it offers. The routine takes two parameters: the service name and the COBOL routine name that should be invoked when a request for the service name is received by the server. Since the service name may differ from the routine name, different service names can be mapped to the same routine.

TPUNADVERTISE allows a server to unadvertise a service that it offers. Even though a particular service may be unadvertised by one server, it may still be offered by others.

Information about service names may be kept in the local configuration. Because each service supports either the request/response or the conversational service paradigm, the local configuration may contain information labeling each service name appropriately.

6.6 Request/Response Service Paradigm

Requests can be issued to services in two ways: synchronously or asynchronously. In both methods, the requester can state whether the request should be sent as part of the caller's current transaction.

6.6.1 Synchronous Request/Response

The TPCALL routine sends a request to the specified service, and returns any response in an application-defined typed record. The call to TPCALL returns after any expected response arrives.

6.6.2 Asynchronous Request/Response

The TPACALL routine also sends a request to the specified service, but it returns without waiting for the service's response, letting the requester do additional work while the service routine processes its request. Using the TPACALL routine allows a requester to exploit parallelism within an application since multiple requests can be simultaneously processed. The TPACALL routine returns to its caller a *communication handle* that is used by the requester to eventually get its reply. If the requester does not require any reply, the requester must indicate that a reply is not expected. However, in this particular case, the request must not be issued in transaction mode.

The TPGETRPLY routine waits to receive a service reply corresponding to a specified request. The routine returns the response in an application-defined typed record.

A requester not wanting to receive a reply from a previously-sent request can call the TPCANCEL routine. This function informs the CRM that any response should be silently discarded. It is worth noting that the TPCANCEL routine does not prevent the service from completing; rather, it relieves the requester from having to receive an unwanted response. It is an error to attempt to cancel a communication handle associated with a global transaction.

6.6.3 Programming Example

See Appendix B for an example of request/response programming in the COBOL programming language.

6.7 Conversational Service Paradigm

In this paradigm, a requester invokes a service routine and converses with it in an application-defined manner. Thus, several messages can be exchanged before the service routine returns ending the conversation. The conversation takes place in a *half-duplex* manner. That is, only one program can send data at a time. Also, the receiver cannot send data until the sender yields it control of the conversation.

The requester initiates conversational communication with a service by calling the TPCONNECT routine. This routine optionally passes application data to the service and specifies which program initially has control of the connection. The requester is returned a communication handle that it uses to refer to the newly established connection during subsequent communication. The routines TPSEND and TPRECV allow APs to exchange data over an open connection.

On the server side of the connection, the CRM listens for and accepts the incoming connection request. The service routine matching the requester's named service is dispatched along with a communication handle that refers to the connection as well as any application data sent as part of the requester's call to TPCONNECT.

A conversational service's communication path with its requester is terminated by the CRM in an orderly manner after the service returns by calling TPRETURN. If the requester wishes to terminate the conversation abortively, rather than orderly, it can call TPDISCON. This routine terminates a connection in a manner that data in transit may be lost and any active transaction associated with that connection is rolled back.

Because communication in the conversational service paradigm is ‘‘longer lived’’ than that of the request/response paradigm, communication *events* that occur during the course of a conversation are reported to either the requester, the service, or both as appropriate. For example, the AP that controls the connection yields control to the receiver by sending it an event. Other events include orderly as well as abortive connection termination.

6.7.1 Programming Example

See Appendix B for an example of conversational programming in the COBOL programming language.

6.8 Transaction Implications

The XATMI interface relies on the **TX** (Transaction Demarcation) interface, published separately, for global transaction demarcation and management. In addition, certain functions in the XATMI interface directly affect the progress of a global transaction.

6.8.1 Transaction Functions Affecting the XATMI Interface

Demarcation

The XATMI interface relies on the following functions of the Transaction Demarcation (TX) interface:

TXBEGIN A demarcation function that indicates that subsequent work performed by the calling AP is in support of a global transaction.

TXCOMMIT A demarcation function that commits all work done on behalf of the current global transaction.

TXROLLBACK A demarcation function that rolls back all work done on behalf of the current global transaction.

The effect of the TX functions on this specification is that an AP detects that the partner's TM has requested completion of the transaction by means of return codes, communication events or errors. The AP may use this information to instruct its TM and its subordinates on how to complete the transaction.

As described in Section 2.2.6 on page 10, APs may generate both request/response and conversational requests. XATMI allows an AP to establish communication requests either inside or outside the boundaries of the global transaction through flags available on the API. Additionally, communication requests established before the global transaction is begun are also not included in the global transaction. The state and validity of these non-transactional requests are not affected by the transaction demarcation (TX) functions. Non-transactional handles may be affected with respect to timeout as described below; however, they are not invalidated by any transaction-related timeouts.

As described above, both request/response and conversational requests generated by the AP may be included in a global transaction if one is active. The handles relating to these communications should be closed, that is terminated normally as described in the reference manual pages, prior to invocation of TXCOMMIT or TXROLLBACK. If such handles are active, that is not closed, at the time TXCOMMIT or TXROLLBACK is invoked, then the handles are invalidated by the TM and the transaction is rolled back. Note that transaction chaining as defined by the transaction demarcation (TX) functions is allowed even though transaction-related XATMI communication handles do not survive transaction boundaries.

Service routines as defined in XATMI may be invoked in transaction mode. In that case, they are subject to the following characteristics with respect to transaction demarcation: TXBEGIN fails with a protocol error since the service routine is already in a transaction; TXCOMMIT and TXROLLBACK fail with a protocol error because they are not the originator of the transaction.

Timeouts

The timeout function of TX also affects the XATMI interface:

TXSETTIMEOUT
: A function that specifies the time interval in which the transaction must complete.

There are two types of timeouts when using XATMI and TX: one is associated with the duration of a transaction from start to finish; the other is associated with the maximum length of time a blocking call remains blocked before the caller regains control. The first kind of timeout is specified when a transaction is started with the TX API's TXBEGIN (see the **TX** (Transaction Demarcation) specification for details). The second kind of timeout can occur when using an XATMI communication routine (for example TPCALL, TPCONNECT or TPRECV). Callers of these routines typically block when awaiting data that has yet to arrive, although they can also block trying to send data (for example, if transmission buffers are full). When the caller is not part of any global (TX) transaction, the maximum amount of time a caller remains blocked is determined in an XATMI provider-specific manner. Routines that return control after either type of timeout has occurred return a particular error code that signifies a timeout event.

Of the two timeout mechanisms, blocking timeouts are performed by default when the caller is not in transaction mode. When a client or server is in transaction mode, it is subject to the timeout value with which the transaction was started and is not subject to any blocking timeout value specified by the XATMI provider.

When a timeout occurs, replies to asynchronous requests may be dropped. That is, if a process is waiting for a particular asynchronous reply and a transaction timeout occurs, the descriptor for that reply becomes invalid and that reply is silently discarded. Similarly, if a transaction timeout occurs during a conversation with a service an event is generated on the associated connection descriptor, that descriptor becomes invalid, and data may be lost. On the other hand, if a blocking timeout occurs, both types of descriptor remain valid and the waiting process can re-issue the call to await the reply.

6.8.2 Effect on Service Calls

Services are either invoked in a global transaction or outside a global transaction. If a requester invokes a service as part of its transaction, the service can participate in only that transaction and the service does not call any transaction demarcation functions. If the client invokes a service outside a transaction, the service routine can originate and complete any number of transactions using the TX (Transaction Demarcation) interface.

In order for a transaction propagated to a service routine to successfully commit, the service routine must first receive all outstanding replies for requests that it generated as well as close any outgoing connections to conversational services that it opened.

6.9 Naming Rules

The XATMI interface uses three kinds of names: *service names*, *buffer type names*, and *buffer sub-type names*. Names are passed in the interface as space-filled **PIC X** fields. Three buffer type names are defined in this specification; other names are application-defined.

Names that meet the following rules are guaranteed to be portable and interoperable across implementations that conform to the XATMI interface.

- A name is composed of one or more characters from the set of letters (A-Z, a-z), digits (0-9), and underscore (_).
- A name must begin with a letter or underscore.
- The case of letters in a name is significant.
- The first 15 characters determine the service name.
- A buffer type name can contain up to 8 characters.
- A buffer sub-type name can contain up to 16 characters.
- A name is terminated by a null (0x00) character or by the first space encountered, or by reaching the length limit for the kind of name.

Chapter 7

COBOL Language Reference Manual Pages

This chapter contains the COBOL language reference manual pages for the XATMI communication API for transaction processing. Following TPINTRO, which describes the COPY files for the XATMI interface, the reference manual pages appear, in alphabetical order, for each COBOL function in the XATMI interface.

NAME

TPINTRO — COPY files for the XATMI interface

DESCRIPTION

The following return codes and setting definitions are used by the COBOL XATMI routines. XATMI interface providers supply these definitions in the four COPY files listed below. Shown for each are the minimum set of record definitions and settings that must be defined in each COPY file.

```
      *
      *  TPSTATUS.cbl
      *
        05 TP-STATUS PIC S9(9) COMP-5.
                88 TPOK                         VALUE 0.
                88 TPEBADDESC                   VALUE 2.
                88 TPEBLOCK                     VALUE 3.
                88 TPEINVAL                     VALUE 4.
                88 TPELIMIT                     VALUE 5.
                88 TPENOENT                     VALUE 6.
                88 TPEOS                        VALUE 7.
                88 TPEPROTO                     VALUE 9.
                88 TPESVCERR                    VALUE 10.
                88 TPESVCFAIL                   VALUE 11.
                88 TPESYSTEM                    VALUE 12.
                88 TPETIME                      VALUE 13.
                88 TPETRAN                      VALUE 14.
                88 TPEGOTSIG                    VALUE 15.
                88 TPEITYPE                     VALUE 17.
                88 TPEOTYPE                     VALUE 18.
                88 TPEEVENT                     VALUE 22.
                88 TPEMATCH                     VALUE 23.

        05 TPEVENT PIC S9(9) COMP-5.
                88 TPEV-NOEVENT                 VALUE 0.
                88 TPEV-DISCONIMM               VALUE 1.
                88 TPEV-SENDONLY                VALUE 2.
                88 TPEV-SVCERR                  VALUE 3.
                88 TPEV-SVCFAIL                 VALUE 4.
                88 TPEV-SVCSUCC                 VALUE 5.

        05 APPL-RETURN-CODE PIC S9(9) COMP-5.
```

The following COBOL record is used whenever sending or receiving application data. **REC-TYPE** indicates the type of data record that is to be sent. **SUB-TYPE** indicates the name of the sub-type for a particular type. **LEN** contains the amount of data to send and the amount received.

```
      *
      *  TPTYPE.cbl
      *
        05 REC-TYPE         PIC X(8).
                88 X-OCTET      VALUE "X_OCTET".
                88 X-COMMON     VALUE "X_COMMON".
        05 SUB-TYPE         PIC X(16).
```

```
        05 LEN              PIC S9(9) COMP-5.
                88 NO-LENGTH    VALUE 0.
        05 TPTYPE-STATUS PIC S9(9) COMP-5.
                88 TPTYPEOK     VALUE 0.
                88 TPTRUNCATE   VALUE 1.
```

The following COBOL record is used by functions to pass settings to and from the communication resource manager.

```
      *
      *  TPSVCDEF.cbl
      *
        05 COMM-HANDLE                  PIC S9(9) COMP-5.
        05 TPBLOCK-FLAG                 PIC S9(9) COMP-5.
                88 TPBLOCK                      VALUE 0.
                88 TPNOBLOCK                    VALUE 1.
        05 TPTRAN-FLAG                  PIC S9(9) COMP-5.
                88 TPTRAN                       VALUE 0.
                88 TPNOTRAN                     VALUE 1.
        05 TPREPLY-FLAG                 PIC S9(9) COMP-5.
                88 TPREPLY                      VALUE 0.
                88 TPNOREPLY                    VALUE 1.
        05 TPTIME-FLAG                  PIC S9(9) COMP-5.
                88 TPTIME                       VALUE 0.
                88 TPNOTIME                     VALUE 1.
        05 TPSIGRSTRT-FLAG              PIC S9(9) COMP-5.
                88 TPNOSIGRSTRT                 VALUE 0.
                88 TPSIGRSTRT                   VALUE 1.
        05 TPGETANY-FLAG                PIC S9(9) COMP-5.
                88 TPGETHANDLE                  VALUE 0.
                88 TPGETANY                     VALUE 1.
        05 TPSENDRECV-FLAG              PIC S9(9) COMP-5.
                88 TPSENDONLY                   VALUE 0.
                88 TPRECVONLY                   VALUE 1.
        05 TPNOCHANGE-FLAG              PIC S9(9) COMP-5.
                88 TPCHANGE                     VALUE 0.
                88 TPNOCHANGE                   VALUE 1.
        05 TPSERVICETYPE-FLAG           PIC S9(9) COMP-5.
                88  TPREQRSP                    VALUE IS 0.
                88  TPCONV                      VALUE IS 1.
        05 SERVICE-NAME                 PIC X(15).
```

The following COBOL record is used by TPRETURN to indicate the status of the transaction.

```
      *
      *  TPSVCRET.cbl
      *
        05 TP-RETURN-VAL PIC S9(9) COMP-5.
                88 TPSUCCESS VALUE 0.
                88 TPFAIL    VALUE 1.
        05 APPL-CODE     PIC S9(9) COMP-5.
```

NAME

TPACALL — send a service request

SYNOPSIS

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY Data record definition.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPACALL" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

DESCRIPTION

TPACALL sends a request message to the service named by **SERVICE-NAME**. *DATA-REC* is the record to be sent and **LEN** specifies the amount of data in *DATA-REC* that should be sent. Note that if *DATA-REC* is a record of a type that does not require a length to be specified, **LEN** is ignored (and may be 0). If *DATA-REC* is a record of a type that does require a length, **LEN** must not be zero. If **REC-TYPE** does not have a subtype, **SUB-TYPE** is ignored (and may be SPACES). If **REC-TYPE** is SPACES, *DATA-REC* and **LEN** are ignored and a request is sent with no data portion. **REC-TYPE** and **SUB-TYPE** must match one of the types and sub-types recognised by **SERVICE-NAME**. Note that for each request sent while in transaction mode, a corresponding reply must ultimately be received.

The valid settings of *TPSVCDEF-REC* are as follows:

TPNOTRAN
: If the caller is in transaction mode and this setting is used, when **SERVICE-NAME** is invoked, it is not performed on behalf of the caller's transaction. If **SERVICE-NAME** does not support transactions, this setting must be used when the caller is in transaction mode. A caller in transaction mode that uses this setting is still subject to the transaction timeout (and no other). If a service fails that was invoked with this setting, the caller's transaction is not affected. Either TPNOTRAN or TPTRAN must be set.

TPTRAN
: If the caller is in transaction mode and this setting is used, when **SERVICE-NAME** is invoked, it is performed on behalf of the caller's transaction. This setting is ignored if the caller is not in transaction mode. Either TPNOTRAN or TPTRAN must be set.

TPNOREPLY
: This setting informs TPACALL that a reply is not expected. When TPNOREPLY is set, the routine returns [TPOK] on success and sets **COMM-HANDLE** to 0, an invalid communication handle. When the caller is in transaction mode, this setting cannot be used when TPTRAN is also set. Either TPNOREPLY or TPREPLY must be set.

TPREPLY
: This setting informs TPACALL that a reply is expected. When TPREPLY is set, the routine returns [TPOK] on success and sets **COMM-HANDLE** to a valid communication handle. When the caller is in transaction mode, this setting must be used when TPTRAN is also set. Either TPNOREPLY or TPREPLY must be set.

TPNOBLOCK
The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Either TPNOBLOCK or TPBLOCK must be set.

TPBLOCK
When TPBLOCK is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either TPNOBLOCK or TPBLOCK must be set.

TPNOTIME
This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either TPNOTIME or TPTIME must be set.

TPTIME
This setting signifies that the caller receives blocking timeouts if a blocking condition exists and the blocking time is reached. Either TPNOTIME or TPTIME must be set.

TPSIGRSTRT
If a signal interrupts any underlying system calls, the interrupted system call is re-issued. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

TPNOSIGRSTRT
If a signal interrupts any underlying system calls, the interrupted system call is not restarted and the routine fails. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

RETURN VALUE

Upon successful completion, TPACALL sets **TP-STATUS** to [TPOK]. In addition, if TPREPLY was set in *TPSVCDEF-REC*, TPACALL returns a valid communication handle in **COMM-HANDLE** that can be used to receive the reply of the request sent.

ERRORS

Under the following conditions, TPACALL fails and sets **TP-STATUS** to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

[TPEINVAL]
Invalid arguments were given (for example, settings in *TPSVCDEF-REC* are invalid).

[TPENOENT]
Cannot send to **SERVICE-NAME** because it does not exist.

[TPEITYPE]
The pair **REC-TYPE** and **SUB-TYPE** is not one of the allowed types and sub-types that **SERVICE-NAME** accepts.

[TPELIMIT]
The caller's request was not sent because the maximum number of outstanding asynchronous requests has been reached.

[TPETRAN]
SERVICE-NAME does not support transactions and TPTRAN was set.

[TPETIME]
A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. If a transaction timeout occurred, any attempts to send new requests or receive outstanding replies fail with [TPETIME] until the transaction has been rolled back.

[TPEBLOCK]
A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
A signal was received and TPNOSIGRSTRT was specified.

[TPEPROTO]
TPACALL was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPCALL, TPCANCEL, TPGETRPLY.

NAME

TPADVERTISE — advertise a service name

SYNOPSIS

```
01  SERVICE-NAME   PIC X(15).

01  PROGRAM-NAME   PIC X(32).

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPADVERTISE" USING SERVICE-NAME PROGRAM-NAME TPSTATUS-REC.
```

DESCRIPTION

TPADVERTISE allows a server to advertise the services that it offers. By default, a server's services are advertised when it is booted and unadvertised when it is shutdown.

TPADVERTISE advertises *SERVICE-NAME* for the server. *SERVICE-NAME* should be 15 characters or fewer, but cannot be SPACES. Longer names are accepted and truncated to 15 characters. Users should make sure that truncated names do not match other service names. *PROGRAM-NAME* is the name of a service program. This program is invoked whenever a request for *SERVICE-NAME* is received by the server. *PROGRAM-NAME* cannot be SPACES.

If *SERVICE-NAME* is already advertised for the server and *PROGRAM-NAME* matches its current program, TPADVERTISE returns success (this includes truncated names that match already advertised names). However, if *SERVICE-NAME* is already advertised for the server but *PROGRAM-NAME* does not match its current program, an error is returned (this can happen if truncated names match already advertised names).

RETURN VALUE

Upon successful completion, TPADVERTISE sets **TP-STATUS** to [TPOK].

ERRORS

Under the following conditions, TPADVERTISE fails and sets **TP-STATUS** to one of the following values:

[TPEINVAL]
: Either *SERVICE-NAME* or *PROGRAM-NAME* is SPACES, or *PROGRAM-NAME* is not the name of a valid program.

[TPELIMIT]
: *SERVICE-NAME* cannot be advertised because of space limitations.

[TPEMATCH]
: *SERVICE-NAME* is already advertised for the server but with a program other than *PROGRAM-NAME*. Although TPADVERTISE fails, *SERVICE-NAME* remains advertised with its current program (that is, *PROGRAM-NAME* does not replace the current program).

[TPEPROTO]
: TPADVERTISE was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPSVCSTART, TPUNADVERTISE.

NAME

TPCALL — send a service request and synchronously await its reply

SYNOPSIS

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 ITPTYPE-REC.
   COPY TPTYPE.

01 IDATA-REC.
   COPY Data record definition.

01 OTPTYPE-REC.
   COPY TPTYPE.

01 ODATA-REC.
   COPY Data record definition.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPCALL" USING TPSVCDEF-REC ITPTYPE-REC IDATA-REC
                    OTPTYPE-REC ODATA-REC TPSTATUS-REC.
```

DESCRIPTION

TPCALL sends a request and synchronously awaits its reply. A call to this routine is the same as calling TPACALL immediately followed by TPGETRPLY. TPCALL sends a request to the service named by **SERVICE-NAME**. The data portion of a request is specified by *IDATA-REC* and **LEN** in *ITPTYPE-REC* specifies how much of *IDATA-REC* to send. Note that if *IDATA-REC* is a record of a type that does not require a length to be specified, **LEN** in *ITPTYPE-REC* is ignored (and may be 0). If *IDATA-REC* is a record of a type that does require a length, **LEN** in *ITPTYPE-REC* must not be zero. If **REC-TYPE** in *ITPTYPE-REC* does not have a subtype, **SUB-TYPE** in *ITPTYPE-REC* is ignored (and may be SPACES). If **REC-TYPE** in *ITPTYPE-REC* is SPACES, *IDATA-REC* and **LEN** in *ITPTYPE-REC* are ignored and a request is sent with no data portion. **REC-TYPE** in *ITPTYPE-REC* and **SUB-TYPE** in *ITPTYPE-REC* must match one of the types and sub-types recognised by **SERVICE-NAME**.

ODATA-REC specifies where the reply is read into, and, on input, **LEN** in *OTPTYPE-REC* indicates the maximum number of bytes that should be moved into *ODATA-REC*. If the same record is to be used for both sending and receiving, *ODATA-REC* should be REDEFINED to *IDATA-REC*. Upon successful return from TPCALL, **LEN** in *OTPTYPE-REC* contains the actual number of bytes moved into *ODATA-REC*. **REC-TYPE** in *OTPTYPE-REC* and **SUB-TYPE** in *OTPTYPE-REC* contain the reply's type and sub-type, respectively. If the reply is larger than *ODATA-REC*, *ODATA-REC* contains only as many bytes as fit in the record. The remainder of the reply is discarded and TPCALL sets TPTRUNCATE.

If **LEN** in *OTPTYPE-REC* is 0 upon successful return, the reply has no data portion and *ODATA-REC* was not modified. It is an error for **LEN** in *OTPTYPE-REC* to be 0 on input.

The valid settings of *TPSVCDEF-REC* are as follows:

TPNOTRAN
: If the caller is in transaction mode and this setting is used, when **SERVICE-NAME** is invoked, it is not performed on behalf of the caller's transaction. If **SERVICE-NAME** does

not support transactions, this setting must be used when the caller is in transaction mode. A caller in transaction mode that uses this setting is still subject to the transaction timeout (and no other). If a service fails that was invoked with this setting, the caller's transaction is not affected. Either TPNOTRAN or TPTRAN must be set.

TPTRAN
: If the caller is in transaction mode and this setting is used, when **SERVICE-NAME** is invoked, it is performed on behalf of the caller's transaction. This setting is ignored if the caller is not in transaction mode. Either TPNOTRAN or TPTRAN must be set.

TPNOCHANGE
: When this setting is used, the type of *ODATA-REC* is not allowed to change. That is, the type and sub-type of the reply record must match **REC-TYPE** in *OTPTYPE-REC* and **SUB-TYPE** in *OTPTYPE-REC*, respectively. Either TPNOCHANGE or TPCHANGE must be set.

TPCHANGE
: The type and/or subtype of the reply record are allowed to differ from those specified in **REC-TYPE** in *OTPTYPE-REC* and **SUB-TYPE** in *OTPTYPE-REC*, respectively, so long as the receiver recognises the incoming record type. Either TPNOCHANGE or TPCHANGE must be set.

TPNOBLOCK
: The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Note that this setting applies only to the send portion of TPCALL: the routine may block waiting for the reply. Either TPNOBLOCK or TPBLOCK must be set.

TPBLOCK
: When TPBLOCK is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either TPNOBLOCK or TPBLOCK must be set.

TPNOTIME
: This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either TPNOTIME or TPTIME must be set.

TPTIME
: This setting signifies that the caller receives blocking timeouts if a blocking condition exists and the blocking time is reached. Either TPNOTIME or TPTIME must be set.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is re-issued. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

TPNOSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is not restarted and the routine fails. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

RETURN VALUE

Upon successful completion, TPCALL sets **TP-STATUS** to [TPOK]. When **TP-STATUS** is set to either [TPOK] or [TPESVCFAIL], **APPL-RETURN-CODE** contains an application-defined value that was sent as part of TPRETURN. If the size of the incoming message is larger than the size specified in **LEN** in OTPTYPE-REC on input, then TPTRUNCATE is set in OTPTYPE-REC and only **LEN** in OTPTYPE-REC bytes are moved into ODATA-REC. The remaining bytes are discarded.

ERRORS

Under the following conditions, TPCALL fails and sets **TP-STATUS** to one of the values below. Unless unless otherwise noted, failure does not affect the caller's transaction, if one exists.

[TPEINVAL]
Invalid arguments were given (for example, settings in *TPSVCDEF-REC* are invalid).

[TPENOENT]
Cannot send to **SERVICE-NAME** because it does not exist.

[TPEITYPE]
The pair **REC-TYPE** in *ITPTYPE-REC* and **SUB-TYPE** in *ITPTYPE-REC* is not one of the allowed types and sub-types that **SERVICE-NAME** accepts.

[TPEOTYPE]
Either the type and sub-type of the reply are not known to the caller, or TPNOCHANGE was set and **REC-TYPE** in *OTPTYPE-REC* and **SUB-TYPE** in *OTPTYPE-REC* do not match the type and sub-type of the reply sent by the service. Neither *ODATA-REC* nor *OTPTYPE-REC* are changed. If the service request was made on behalf of the caller's current transaction, the transaction is marked rollback-only since the reply is discarded.

[TPETRAN]
SERVICE-NAME does not support transactions and TPTRAN was set.

[TPETIME]
A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. In either case, neither *ODATA-REC* nor *OTPTYPE-REC* are changed. If a transaction timeout occurred, any attempts to send new requests or receive outstanding replies fail with [TPETIME] until the transaction has been rolled back.

[TPESVCFAIL]
The service routine sending the caller's reply called TPRETURN with TPFAIL. This is an application-level failure. The contents of the service's reply, if one was sent, are available in *ODATA-REC*. If the service request was made on behalf of the caller's current transaction, the transaction is marked rollback-only. Note that so long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Such attempts should be made with TPNOTRAN set if they are to have any lasting effect. Any work performed on behalf of the caller's transaction is rolled back upon transaction completion.

[TPESVCERR]
An error was encountered either in invoking a service routine or during its completion in TPRETURN (for example, bad arguments were passed). No reply data is returned when this error occurs (that is, neither *ODATA-REC* nor *OTPTYPE-REC* are changed). If the service request was made on behalf of the caller's transaction, the transaction is marked rollback-only. Note that so long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Such attempts should be made with TPNOTRAN set if they are to have any lasting effect. Any work performed on behalf of the caller's transaction is rolled back upon transaction completion.

[TPEBLOCK]
A blocking condition was found on the send portion of TPCALL and TPNOBLOCK was specified.

[TPGOTSIG]
A signal was received and TPNOSIGRSTRT was specified.

[TPEPROTO]
TPCALL was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPACALL, TPGETRPLY, TPRETURN.

NAME

TPCANCEL — cancel a communication handle for an outstanding reply

SYNOPSIS

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPCANCEL" USING TPSVCDEF-REC TPSTATUS-REC.
```

DESCRIPTION

TPCANCEL cancels a communication handle, **COMM-HANDLE**, returned by TPACALL. It is an error to attempt to cancel a communication handle associated with a global transaction.

Upon success, **COMM-HANDLE** is no longer valid and any reply received (by the communication resource manager) on behalf of **COMM-HANDLE** is silently discarded.

RETURN VALUE

Upon successful completion, TPCANCEL sets **TP-STATUS** to [TPOK].

ERRORS

Under the following conditions, TPCANCEL fails and sets **TP-STATUS** to one of the following values:

[TPEBADDESC]
: **COMM-HANDLE** contains an invalid communication handle.

[TPETRAN]
: **COMM-HANDLE** is associated with the caller's global transaction. **COMM-HANDLE** remains valid and the caller's current transaction is not affected.

[TPEPROTO]
: TPCANCEL was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPACALL.

NAME

TPCONNECT — establish a conversational service connection

SYNOPSIS

```
01  TPSVCDEF-REC.
    COPY TPSVCDEF.

01  TPTYPE-REC.
    COPY TPTYPE.

01  DATA-REC.
    COPY Data record definition.

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPCONNECT" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

DESCRIPTION

TPCONNECT allows a program to set up a half-duplex connection to a conversational service, **SERVICE-NAME**.

As part of setting up a connection, the caller can pass application-defined data to the receiving service routine. If the caller chooses to pass data, *DATA-REC* contains the data and **LEN** specifies how much of the record to send. Note that if *DATA-REC* is a record of a type that does not require a length to be specified, **LEN** is ignored (and may be 0). If *DATA-REC* is a record of a type that does require a length, **LEN** must not be zero. If **REC-TYPE** does not have a subtype, **SUB-TYPE** is ignored (and may be SPACES). If **REC-TYPE** is SPACES, *DATA-REC* and **LEN** are ignored (no application data is passed to the conversational service). **REC-TYPE** and **SUB-TYPE** must match one of the types and sub-types recognised by **SERVICE-NAME**.

Because the conversational service receives *DATA-REC* and **LEN** upon successful return from TPSVCSTART, the service does not call TPRECV to get the data sent by TPCONNECT.

The valid settings of *TPSVCDEF-REC* are as follows:

TPNOTRAN
: If the caller is in transaction mode and this setting is used, when **SERVICE-NAME** is invoked, it is not performed on behalf of the caller's transaction. If **SERVICE-NAME** does not support transactions, this setting must be used when the caller is in transaction mode. A caller in transaction mode that uses this setting is still subject to the transaction timeout (and no other). If a service fails that was invoked with this setting, the caller's transaction is not affected. Either TPNOTRAN or TPTRAN must be set.

TPTRAN
: If the caller is in transaction mode and this setting is used, when **SERVICE-NAME** is invoked, it is performed on behalf of the caller's transaction. This setting is ignored if the caller is not in transaction mode. Either TPNOTRAN or TPTRAN must be set.

TPSENDONLY
: The caller wants the connection to be set up initially such that it can send data and the called service can only receive data (that is, the caller initially has control of the connection). Either TPSENDONLY or TPRECVONLY must be specified.

TPRECVONLY
: The caller wants the connection to be set up initially such that it can only receive data and the called service can send data (that is, the service being called initially has control of the connection). Either TPSENDONLY or TPRECVONLY must be specified.

TPNOBLOCK
: The connection is not established and the data is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Either TPNOBLOCK or TPBLOCK must be set.

TPBLOCK
: When TPBLOCK is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either TPNOBLOCK or TPBLOCK must be set.

TPNOTIME
: This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either TPNOTIME or TPTIME must be set.

TPTIME
: This setting signifies that the caller receives blocking timeouts if a blocking condition exists and the blocking time is reached. Either TPNOTIME or TPTIME must be set.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is re-issued. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

TPNOSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is not restarted and the routine fails. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

RETURN VALUE

Upon successful completion, TPCONNECT sets **TP-STATUS** to [TPOK] and returns a valid communication handle in **COMM-HANDLE** that is used to refer to the connection in subsequent calls.

ERRORS

Under the following conditions, TPCONNECT fails and sets **TP-STATUS** to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

[TPEINVAL]
: Invalid arguments were given (for example, settings in *TPSVCDEF-REC* are invalid).

[TPENOENT]
: Cannot initiate a connection to **SERVICE-NAME** because it does not exist.

[TPEITYPE]
: The pair **REC-TYPE** and **SUB-TYPE** is not one of the allowed types and sub-types that **SERVICE-NAME** accepts.

[TPELIMIT]
: The connection was not established because the maximum number of outstanding connections has been reached.

[TPETRAN]
: **SERVICE-NAME** does not support transactions and TPTRAN was set.

[TPETIME]
A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. If a transaction timeout occurred, any attempts to send or receive messages on any connections or to start a new connection fail with [TPETIME] until the transaction has been rolled back.

[TPEBLOCK]
A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
A signal was received and TPNOSIGRSTRT was specified.

[TPEPROTO]
TPCONNECT was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPDISCON, TPRECV, TPSEND, TPSVCSTART.

NAME

TPDISCON — terminate a conversational service connection abortively

SYNOPSIS

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPDISCON" USING TPSVCDEF-REC TPSTATUS-REC.
```

DESCRIPTION

TPDISCON immediately terminates the connection specified by **COMM-HANDLE** and generates a TPEV-DISCONIMM event on the other end of the connection.

TPDISCON can be called only by the initiator of the conversation. TPDISCON cannot be called within a conversational service on the communication handle with which it was invoked. Rather, a conversational service must use TPRETURN to signify that it has completed its part of the conversation. Similarly, even though a program communicating with a conversational service can issue TPDISCON, the preferred way is to let the service terminate the connection in TPRETURN; doing so ensures correct results.

TPDISCON causes the connection to be terminated immediately (that is, abortively rather than orderly). Any data that has not yet reached its destination may be lost. TPDISCON can be issued even when the program on the other end of the connection is participating in the caller's transaction. In this case, the transaction must be rolled back. Also, the caller does not need to have control of the connection when TPDISCON is called.

RETURN VALUE

Upon successful completion, TPDISCON sets **TP-STATUS** to [TPOK].

ERRORS

Under the following conditions, TPDISCON fails and sets **TP-STATUS** to one of the following values:

[TPEBADDESC]
: Either **COMM-HANDLE** is invalid or it is the communication handle with which a conversational service was invoked.

[TPETIME]
: A timeout occurred. The communication handle is no longer valid.

[TPEPROTO]
: TPDISCON was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPCONNECT, TPRECV, TPRETURN, TPSEND.

NAME

TPGETRPLY — get a reply from a previous service request

SYNOPSIS

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY Data record definition.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPGETRPLY" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

DESCRIPTION

TPGETRPLY returns a reply from a previously sent request. TPGETRPLY either returns a reply for a particular request, or it returns any reply that is available. Both options are described below.

DATA-REC specifies where the reply is read into, and, on input, **LEN** indicates the maximum number of bytes that should be moved into *DATA-REC*. Upon successful return from TPGETRPLY, **LEN** contains the actual number of bytes moved into *DATA-REC*. **REC-TYPE** and **SUB-TYPE** contain the data's type and sub-type, respectively. If the reply is larger than *DATA-REC*, *DATA-REC* contain only as many bytes as fit in the record. The remainder of the reply is discarded and TPGETRPLY sets TPTRUNCATE.

If **LEN** is 0 upon successful return, the reply has no data portion and *DATA-REC* was not modified. It is an error for **LEN** to be 0 on input.

The valid settings of *TPSVCDEF-REC* are as follows:

TPGETANY
: This setting signifies that TPGETRPLY should ignore the communication handle indicated by **COMM-HANDLE** on input, return any reply available, and set **COMM-HANDLE** on output to the communication handle for the reply returned. If no replies exist, TPGETRPLY can optionally wait for one to arrive. Either TPGETANY or TPGETHANDLE must be set.

TPGETHANDLE
: This setting signifies that TPGETRPLY should use the communication handle identified by **COMM-HANDLE** on input and return a reply available for that handle only. If no replies exist, TPGETRPLY can optionally wait for one to arrive. Either TPGETANY or TPGETHANDLE must be set.

TPNOCHANGE
: When this setting is used, the type of *DATA-REC* is not allowed to change. That is, the type and sub-type of the reply record must match **REC-TYPE** and **SUB-TYPE**, respectively. Either TPNOCHANGE or TPCHANGE must be set.

TPCHANGE
: The type and/or subtype of the reply record are allowed to differ from those specified in **REC-TYPE** and **SUB-TYPE**, respectively, so long as the receiver recognises the incoming record type. Either TPNOCHANGE or TPCHANGE must be set.

TPNOBLOCK
: TPGETRPLY does not wait for the reply to arrive. If a reply is available, TPGETRPLY gets the reply and returns. Either TPNOBLOCK or TPBLOCK must be set.

TPBLOCK
: When TPBLOCK is specified and no reply is available, the caller blocks until the reply arrives or a timeout occurs (either transaction or blocking timeout). Either TPNOBLOCK or TPBLOCK must be set.

TPNOTIME
: This setting signifies that the caller is willing to block indefinitely for its reply and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either TPNOTIME or TPTIME must be set.

TPTIME
: This setting signifies that the caller receives blocking timeouts if a blocking condition exists and the blocking time is reached. Either TPNOTIME or TPTIME must be set.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is re-issued. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

TPNOSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is not restarted and the routine fails. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

Except as noted below, **COMM-HANDLE** is no longer valid after its reply is received.

RETURN VALUE

Upon successful completion, TPGETRPLY sets **TP-STATUS** to [TPOK]. When **TP-STATUS** is set to either [TPOK] or [TPESVCFAIL], **APPL-RETURN-CODE** contains an application-defined value that was sent as part of TPRETURN. If the size of the incoming message is larger than the size specified in **LEN** on input, TPTRUNCATE is set and only **LEN** bytes are moved into DATA-REC. The remaining bytes are discarded.

ERRORS

Under the following conditions, TPGETRPLY fails and sets **TP-STATUS** as indicated below. Note that if TPGETHANDLE is set, **COMM-HANDLE** is invalidated unless otherwise stated. If TPGETANY is set, **COMM-HANDLE** identifies the descriptor for the reply on which the failure occurred; if an error occurred before a reply could be retrieved, **COMM-HANDLE** is set to 0, unless otherwise stated. Also, the failure does not affect the caller's transaction, if one exists, unless otherwise stated.

[TPEINVAL]
: Invalid arguments were given (for example, settings in *TPSVCDEF-REC* are invalid).

[TPEBADDESC]
: **COMM-HANDLE** contains an invalid communication handle.

[TPEOTYPE]
: Either the type and sub-type of the reply are not known to the caller, or TPNOCHANGE was set and **REC-TYPE** and **SUB-TYPE** do not match the type and sub-type of the reply sent by the service. Neither *DATA-REC* nor *TPTYPE-REC* are changed. If the reply was to be received on behalf of the caller's current transaction, the transaction is marked rollback-only since the reply is discarded.

[TPETIME]
A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. In either case, neither *DATA-REC* nor *TPTYPE-REC* are changed. If TPGETHANDLE was set, **COMM-HANDLE** remains valid unless the caller is in transaction mode. If a transaction timeout occurred, any attempts to send new requests or receive outstanding replies fail with [TPETIME] until the transaction has been rolled back.

[TPESVCFAIL]
The service routine sending the caller's reply called TPRETURN with TPFAIL. This is an application-level failure. The contents of the service's reply, if one was sent, are available in *DATA-REC*. If the reply was received on behalf of the caller's transaction, the transaction is marked rollback-only. Note that so long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Such attempts should be made with TPNOTRAN set if they are to have any lasting effect. Any work performed on behalf of the caller's transaction is rolled back upon transaction completion.

[TPESVCERR]
An error was encountered either in invoking a service routine or during its completion in TPRETURN (for example, bad arguments were passed). No reply data is returned when this error occurs (that is, neither *DATA-REC* nor *TPTYPE-REC* are changed). If the reply was received on behalf of the caller's transaction, the transaction is marked rollback-only. Note that so long as the transaction has not timed out, further communication may be attempted before rolling back the transaction. Such attempts may be processed normally or may fail (producing an error return or event). Such attempts should be made with TPNOTRAN set if they are to have any lasting effect. Any work performed on behalf of the caller's transaction is rolled back upon transaction completion.

[TPEBLOCK]
A blocking condition exists and TPNOBLOCK was specified. **COMM-HANDLE** remains valid.

[TPGOTSIG]
A signal was received and TPNOSIGRSTRT was specified.

[TPEPROTO]
TPGETRPLY was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPACALL, TPCANCEL, TPRETURN.

NAME

TPRECV — receive a message in a conversational connection

SYNOPSIS

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY Data record definition.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPRECV" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

DESCRIPTION

TPRECV is used to receive data sent across an open connection from another program. **COMM-HANDLE** specifies on which open connection to receive data. **COMM-HANDLE** is a communication handle returned from either TPCONNECT or TPSVCSTART. *DATA-REC* specifies where the message is read into, and, on input, **LEN** indicates the maximum number of bytes that should be moved into *DATA-REC*.

Upon successful return, and for several event types, **LEN** contains the actual number of bytes moved into *DATA-REC*. **REC-TYPE** and **SUB-TYPE** contain the data's type and sub-type, respectively. If the message is larger than *DATA-REC*, *DATA-REC* contains only as many bytes as fit in the record. The remainder of the message is discarded and TPRECV sets TPTRUNCATE.

If **LEN** is 0 upon successful return, the message has no data portion and *DATA-REC* was not modified. It is an error for **LEN** to be 0 on input.

TPRECV can be issued only by the program that does not have control of the connection.

The valid settings of *TPSVCDEF-REC* are as follows:

TPNOCHANGE
: When this setting is used, the type of *DATA-REC* is not allowed to change. That is, the type and sub-type of the message received must match **REC-TYPE** and **SUB-TYPE**, respectively. Either TPNOCHANGE or TPCHANGE must be set.

TPCHANGE
: The type or subtype of the message received is allowed to differ from those specified in **REC-TYPE** and **SUB-TYPE**, respectively, so long as the receiver recognises the incoming record type. Either TPNOCHANGE or TPCHANGE must be set.

TPNOBLOCK
: TPRECV does not wait for data to arrive. If data is already available to receive, TPRECV gets the data and returns. Either TPNOBLOCK or TPBLOCK must be set.

TPBLOCK
: When TPBLOCK is specified and no data is available to receive, the caller blocks until data arrives. Either TPNOBLOCK or TPBLOCK must be set.

TPNOTIME
This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either TPNOTIME or TPTIME must be set.

TPTIME
This setting signifies that the caller receives blocking timeouts if a blocking condition exists and the blocking time is reached. Either TPNOTIME or TPTIME must be set.

TPSIGRSTRT
If a signal interrupts any underlying system calls, the interrupted system call is re-issued. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

TPNOSIGRSTRT
If a signal interrupts any underlying system calls, the interrupted system call is not restarted and the routine fails. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

If an event exists for **COMM-HANDLE**, and TPRECV encounters no errors, TPRECV returns setting **TP-STATUS** to [TPEEVENT]. The event type is returned in **TP-EVENT**. Data can be received along with the TPEV-SVCSUCC, TPEV-SVCFAIL, and TPEV-SENDONLY events. Valid events for TPRECV are as follows:

TPEV-DISCONIMM
Received by the subordinate of a conversation, this event indicates that the originator of the conversation has either issued an immediate disconnect on the connection via TPDISCON, or it issued TPRETURN, TXCOMMIT or TXROLLBACK with the connection still open. This event is also returned to the originator or subordinate when a connection is broken due to a communication error (for example, a server, machine, or network failure). Because this is an immediate disconnection notification (that is, abortive rather than orderly), data in transit may be lost. If the two programs were participating in the same transaction, the transaction is marked rollback-only. **COMM-HANDLE** is no longer valid.

TPEV-SENDONLY
The program on the other end of the connection has relinquished control of the connection. The recipient of this event is allowed to send data but cannot receive any data until it relinquishes control.

TPEV-SVCERR
Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued TPRETURN. TPRETURN encountered an error that precluded the service from returning successfully. For example, bad arguments may have been passed to TPRETURN or it may have been called while the service had open connections to other subordinates. Due to the nature of this event, any application-defined data or return code is not available. The connection has been terminated and **COMM-HANDLE** is no longer valid. If this event occurred as part of the recipient's transaction, the transaction is marked rollback-only.

TPEV-SVCFAIL
Received by the originator of a conversation, this event indicates that the subordinate service on the other end of the conversation has finished unsuccessfully as defined by the application (that is, it called TPRETURN with TPFAIL). If the subordinate service was in control of this connection when TPRETURN was called, it can pass a record back to the originator of the connection. As part of ending the service routine, the server has terminated the connection. Thus, **COMM-HANDLE** is no longer valid. If this event occurred as part of the recipient's transaction, the transaction is marked rollback-only.

TPEV-SVCSUCC

Received by the originator of a conversation, this event indicates that the subordinate service on the other end of the conversation has finished successfully as defined by the application (that is, it called TPRETURN with TPSUCCESS). As part of ending the service routine, the server has terminated the connection. Thus, **COMM-HANDLE** is no longer valid. If the recipient is in transaction mode, it can either commit (if it is also the initiator) or roll back the transaction causing the work done by the server (if also in transaction mode) to either commit or roll back.

RETURN VALUE

Upon successful completion, TPRECV sets **TP-STATUS** to [TPOK]. If an event exists and no errors were encountered, TPRECV sets **TP-STATUS** to [TPEEVENT]. When **TP-STATUS** is set to [TPEEVENT] and **TP-EVENT** is either TPEV-SVCSUCC or TPEV-SVCFAIL, **APPL-RETURN-CODE** contains an application-defined value that was sent as part of TPRETURN. If the size of the incoming message is larger than the size specified in **LEN** on input, TPTRUNCATE is set and only **LEN** bytes are moved into DATA-REC. The remaining bytes are discarded.

ERRORS

Under the following conditions, TPRECV fails and sets **TP-STATUS** to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

[TPEINVAL]

Invalid arguments were given (for example, settings in *TPSVCDEF-REC* are invalid).

[TPEBADDESC]

COMM-HANDLE contains an invalid communication handle.

[TPEOTYPE]

Either the type and sub-type of the incoming message are not known to the caller, or TPNOCHANGE was set and **REC-TYPE** and **SUB-TYPE** do not match the type and sub-type of the incoming message. If the conversation is part of the caller's current transaction, the transaction is marked rollback-only since the incoming message is discarded. When this error occurs, any event for **COMM-HANDLE** is dropped and the conversation may now be in an indeterminate state. The caller should terminate the conversation.

[TPETIME]

A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. In either case, neither *DATA-REC* nor *TPTYPE-REC* are changed. If a transaction timeout occurred, any attempts to send or receive messages on any connections or to start a new connection fail with [TPETIME] until the transaction has been rolled back.

[TPEEVENT]

An event occurred and its type is available in **TP-EVENT**.

[TPEBLOCK]

A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]

A signal was received and TPNOSIGRSTRT was specified.

[TPEPROTO]

TPRECV was called in an improper context.

[TPESYSTEM]

A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]

An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPCONNECT, TPDISCON, TPSEND.

NAME

TPRETURN — return from a service routine

SYNOPSIS

```
01 TPSVCRET-REC.
   COPY TPSVCRET.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY Data record definition.

COPY TPRETURN [REPLACING TPSVCRET-REC BY TPSVCRET-REC]
              [REPLACING TPTYPE-REC BY TPTYPE-REC]
              [REPLACING DATA-REC BY DATA-REC].
```

DESCRIPTION

TPRETURN indicates that a service routine has completed. TPRETURN is a file containing the last sequence of COBOL code to be executed in the service. It contains references to three data record names: TPSVCRET-REC, TPTYPE-REC and DATA-REC that may be substituted by the record names effectively used in the service routine. Since TPRETURN contains an **EXIT PROGRAM** statement, it should be issued in the same routine that was invoked by the communication resource manager so that control can be returned to the communication resource manager (that is, TPRETURN should not be invoked in a sub-program of the service routine since control would not return to the communication resource manager).

TPRETURN is used to send a service's reply message. If the program receiving the reply is waiting in TPCALL, TPGETRPLY or TPRECV, after a successful call to TPRETURN, the reply is available in the receiver's record.

For conversational services, TPRETURN also terminates the connection. That is, the service routine cannot call TPDISCON directly. To ensure correct results, the program that connected to the conversational service should not call TPDISCON; rather, it should wait for notification that the conversational service has completed (that is, it should wait for one of the events, like TPEV-SVCSUCC or TPEV-SVCFAIL, sent by TPRETURN).

If the service routine was in transaction mode, TPRETURN places the service's portion of the transaction in a state where it may be either committed or rolled back when the transaction is completed. A service may be invoked multiple times as part of the same transaction so it is not necessarily fully committed nor rolled back until either TXCOMMIT or TXROLLBACK is called by the originator of the transaction.

TPRETURN should be called after receiving all replies expected from service requests initiated by the service routine. Otherwise, depending on the nature of the service, either a [TPESVCERR] error or a TPEV-SVCERR event are returned to the program that initiated communication with the service routine. Any outstanding replies that are not received are automatically dropped by the communication resource manager. In addition, the communication handles for those replies become invalid.

TPRETURN should be called after closing all connections initiated by the service. Otherwise, depending on the nature of the service, either a [TPESVCERR] or a TPEV-SVCERR event is returned to the program that initiated communication with the service routine. Also, an immediate disconnect event (that is, TPEV-DISCONIMM) is sent over all open connections to subordinates.

Concerning control of the connection, if the service routine does not have control over the connection with which it was invoked when it issues TPRETURN, two outcomes are possible. Firstly, if the service routine calls TPRETURN with **TP-RETURN-VAL** (in *TPSVCRET-REC*) set to TPFAIL and **REC-TYPE** (in *TPTYPE-REC*) set to SPACES (that is, no data is sent), a TPEV-SVCFAIL event is sent to the originator of this conversation. Secondly, if any other invocation of TPRETURN is used, a TPEV-SVCERR event is sent to the originator.

Since a conversational service has only one open connection that it did not initiate, the communication resource manager knows over which communication handle data (and any event) should be sent. For this reason, a communication handle is not passed to TPRETURN.

The following is a description of TPRETURN's arguments. **TP-RETURN-VAL** can be set to one of the following.

TPSUCCESS
: The service has terminated successfully. If data is present, it is sent (barring any failures processing the return). If the caller is in transaction mode, TPRETURN places the caller's portion of the transaction in a state such that it can be committed when the transaction ultimately commits. Note that a call to TPRETURN does not necessarily finalise an entire transaction. Also, even though the caller indicates success, if there are any outstanding replies or open connections to subordinates, or if any work done within the service caused its transaction to be marked rollback-only, a failed message is sent (that is, the recipient of the reply receives a [TPESVCERR] indication or a TPEV-SVCERR event). Note that if a transaction becomes rollback-only while in the service routine for any reason, **TP-RETURN-VAL** should be set to TPFAIL. If TPSUCCESS is specified for a conversational service, a TPEV-SVCSUCC event is generated.

TPFAIL
: The service has terminated unsuccessfully from an application standpoint. An error is reported to the program receiving the reply. That is, the call to get the reply has failed and the recipient receives a [TPSVCFAIL] indication or a TPEV-SVCFAIL event. If the caller is in transaction mode, TPRETURN marks the transaction as rollback-only (note that the transaction may already be marked rollback-only). Barring any failures in processing the return, the caller's data is sent, if present. One reason for not sending the caller's data is when a transaction timeout has occurred. In this case, the program waiting for the reply receives an error of [TPETIME].

If **TP-RETURN-VAL** does not contain one of these two values, TPFAIL is assumed.

An application-defined return code, **APPL-CODE** (in *TPSVCRET-REC*), may be sent to the program receiving the service reply. This code is sent regardless of the setting of **TP-RETURN-VAL** as long as a reply can be successfully sent (that is, as long as the receiving call returns success or [TPESVCFAIL], or receives one of the events TPEV-SVCSUCC or TPEV-SVCFAIL). The value of **APPL-CODE** is available to the receiver in **APPL-RETURN-CODE** in *TPSTATUS-REC*.

DATA-REC is the record to be sent and **LEN** (in *TPTYPE-REC*) specifies the amount of data in *DATA-REC* that should be sent. Note that if *DATA-REC* is a record of a type that does not require a length to be specified, **LEN** is ignored (and may be 0). If *DATA-REC* is a record of a type that does require a length, **LEN** must not be zero. If **REC-TYPE** does not have a subtype, **SUB-TYPE** is ignored (and may be SPACES). If **REC-TYPE** is SPACES, *DATA-REC* and **LEN** are ignored. In this case, if a reply is expected by the program that invoked the service, a reply is sent with no data portion. If no reply is expected, TPRETURN ignores any data passed to it and returns sending no reply.

If the service is conversational, there are two cases where the data record is not transmitted:

- If the connection has already been terminated when the call is made (that is, the caller has received TPEV-DISCONIMM on the connection), this call simply ends the service routine and rolls back the current transaction, if one exists. In this case, the caller's data record cannot be transmitted.
- If the caller does not have control of the connection, either TPEV-SVCFAIL or TPEV-SVCERR is sent to the originator of the connection as described above. Regardless of which event the originator receives, no data record is transmitted; however, if the originator receives the TPEV-SVCFAIL event, the return code is available in the originator's **APPL-RETURN-CODE** in *TPSTATUS-REC*.

RETURN VALUE

Since TPRETURN contains an **EXIT PROGRAM** statement, no value is returned to the caller, nor does control return to the service routine. If a service routine returns without using TPRETURN (that is, it uses an **EXIT PROGRAM** statement directly or ‘‘falls out of the service routine’’), the server returns a service error to the service requester. In addition, all open connections to subordinates are disconnected immediately, and any outstanding asynchronous replies are dropped. If the server was in transaction mode at the time of failure, the transaction is marked rollback-only. Note also that if TPRETURN is used outside a service routine (that is, by routines that are not services), it returns having no effect.

ERRORS

Since TPRETURN ends the service routine, any errors encountered either in handling arguments or in processing cannot be indicated to the routine's caller. Such errors cause **TP-STATUS** to be set to [TPESVCERR] for a program receiving the service's outcome via either TPCALL or TPGETRPLY, and cause the event, TPEV-SVCERR, to be sent over the conversation to a program using TPSEND or TPRECV.

SEE ALSO

TPCALL, TPCONNECT, TPDISCON, TPSEND, TPSVCSTART.

NAME

TPSEND — send a message in a conversational connection

SYNOPSIS

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY Data record definition.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPSEND" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

DESCRIPTION

TPSEND is used to send data across an open connection to another program. The caller must have control of the connection. **COMM-HANDLE** specifies the open connection over which data is sent. **COMM-HANDLE** is a communication handle returned from either TPCONNECT or TPSVCSTART.

DATA-REC contains the data to be sent and **LEN** specifies how much of the data to send. Note that if *DATA-REC* is a record of a type that does not require a length to be specified, **LEN** is ignored (and may be 0). If *DATA-REC* is a record of a type that does require a length, **LEN** must not be zero. If **REC-TYPE** does not have a subtype, **SUB-TYPE** is ignored (and may be SPACES). If **REC-TYPE** is SPACES, *DATA-REC* and **LEN** are ignored and a message is sent with no data (this might be done, for instance, to grant control of the connection without transmitting any data).

The valid settings of *TPSVCDEF-REC* are as follows:

TPRECVONLY
: This setting signifies that, after the caller's data is sent, the caller gives up control of the connection (that is, the caller cannot issue any more TPSEND calls). When the receiver on the other end of the connection receives the data sent by TPSEND, it also receives an event (TPEV-SENDONLY) indicating that it has control of the connection (and cannot issue any more TPRECV calls). Either TPRECVONLY or TPSENDONLY must be set.

TPSENDONLY
: This setting signifies that the caller wants to remain in control of the connection. Either TPRECVONLY or TPSENDONLY must be set.

TPNOBLOCK
: The data and any events are not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Either TPNOBLOCK or TPBLOCK must be set.

TPBLOCK
: When TPBLOCK is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either TPNOBLOCK or TPBLOCK must be set.

TPNOTIME
: This setting signifies that the caller is willing to block indefinitely and wants to be immune

to blocking timeouts. Transaction timeouts may still occur. Either TPNOTIME or TPTIME must be set.

TPTIME
: This setting signifies that the caller receives blocking timeouts if a blocking condition exists and the blocking time is reached. Either TPNOTIME or TPTIME must be set.

TPSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is re-issued. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

TPNOSIGRSTRT
: If a signal interrupts any underlying system calls, the interrupted system call is not restarted and the routine fails. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

If an event exists for **COMM-HANDLE**, TPSEND returns without sending the caller's data. The event type is returned in **TP-EVENT**. Valid events for TPSEND are as follows:

TPEV-DISCONIMM
: Received by the subordinate of a conversation, this event indicates that the originator of the conversation has either issued an immediate disconnect on the connection via TPDISCON, or it issued TPRETURN, TXCOMMIT or TXROLLBACK with the connection still open. This event is also returned to the originator or subordinate when a connection is broken due to a communication error (for example, a server, machine, or network failure).

TPEV-SVCERR
: Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued TPRETURN without having control of the conversation. In addition, TPRETURN was issued in a manner different from that described for TPEV-SVCFAIL below.

TPEV-SVCFAIL
: Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued TPRETURN without having control of the conversation. In addition, TPRETURN was issued with the command TPFAIL and no data record (that is, the **REC-TYPE** passed to TPRETURN was set to SPACES).

Because each of these events indicates an immediate disconnection notification (that is, abortive rather than orderly), data in transit may be lost. The communication handle used for the connection is no longer valid. If the two programs were participating in the same transaction, the transaction has been marked rollback-only.

RETURN VALUE

Upon successful completion, TPSEND sets **TP-STATUS** to [TPOK]. When **TP-STATUS** is set to [TPEEVENT] and **TP-EVENT** is TPEV-SVCFAIL, **APPL-RETURN-CODE** contains an application-defined value that was sent as part of TPRETURN.

ERRORS

Under the following conditions, TPSEND fails and sets **TP-STATUS** to one of the values below. Unless otherwise noted, failure does not affect the caller's transaction, if one exists.

[TPEINVAL]
: Invalid arguments were given (for example, settings in *TPSVCDEF-REC* are invalid).

[TPEBADDESC]
: **COMM-HANDLE** contains an invalid communication handle.

[TPETIME]
A timeout occurred. If the caller is in transaction mode, a transaction timeout occurred and the transaction is marked rollback-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. In either case, neither *DATA-REC* nor *TPTYPE-REC* are changed. If a transaction timeout occurred, any attempts to send or receive messages on any connections or to start a new connection fail with [TPETIME] until the transaction has been rolled back.

[TPEEVENT]
An event occurred and its type is available in **TP-EVENT**. *DATA-REC* is not sent when this error occurs.

[TPEBLOCK]
A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
A signal was received and TPNOSIGRSTRT was specified.

[TPEPROTO]
TPSEND was called in an improper context.

[TPESYSTEM]
A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPCONNECT, TPDISCON, TPRECV, TPRETURN.

NAME

TPSVCSTART — start a service routine

SYNOPSIS

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY Data record definition.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPSVCSTART" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

DESCRIPTION

TPSVCSTART is the first routine called when writing a service routine. In fact, it is an error to issue any other XATMI call within a service routine before calling TPSVCSTART. TPSVCSTART is used to retrieve the service's parameters and data. This routine is used for services that receive requests via TPCALL or TPACALL routines as well as by services that communicate via TPCONNECT, TPSEND and TPRECV routines.

Service routines processing requests made via either TPCALL or TPACALL receive, at most, one incoming message (upon successfully returning from TPSVCSTART) and send, at most, one reply (upon exiting the service routine with TPRETURN).

Conversational services, on the other hand, are invoked by connection requests with, at most, one incoming message along with a means of referring to the open connection. Upon successfully returning from TPSVCSTART, either the connecting program or the conversational service may send and receive data as defined by the application. The connection is half-duplex in nature meaning that one side controls the conversation (that is, it sends data) until it explicitly gives up control to the other side of the connection.

Concerning transactions, service routines can participate in, at most, one transaction if invoked in transaction mode. As far as the service routine writer is concerned, the transaction ends upon returning from the service routine. If the service routine is not invoked in transaction mode, the service routine may originate as many transactions as it wants using TXBEGIN, TXCOMMIT and TXROLLBACK. Note that TPRETURN is not used to complete a transaction. Thus, it is an error to call TPRETURN with an outstanding transaction that originated within the service routine.

DATA-REC specifies where the service's data is read into, and, on input, **LEN** indicates the maximum number of bytes that should be moved into *DATA-REC*. Upon successful return from TPSVCSTART, **LEN** contains the actual number of bytes moved into *DATA-REC*. **REC-TYPE** and **SUB-TYPE** contain the data's type and sub-type, respectively. If the message is larger than *DATA-REC*, *DATA-REC* contains only as many bytes as will fit in the record. The remainder of the message is discarded and TPSVCSTART sets TPTRUNCATE.

If **LEN** is 0 upon successful return, the service has no incoming data and *DATA-REC* was not modified. It is an error for **LEN** to be 0 on input.

Upon successful return, **SERVICE-NAME** is populated with the service name that the requesting program used to invoke the service.

The possible settings of *TPSVCDEF-REC* upon the return of TPSVCSTART are as follows:

TPREQRSP
: The service was invoked with either TPCALL or TPACALL. This setting is mutually exclusive with TPCONV.

TPCONV
: The service was invoked with TPCONNECT. The communication handle for the conversation is available in **COMM-HANDLE**. This setting is mutually exclusive with TPREQRSP.

TPNOTRAN
: The service routine is not in transaction mode. This setting is mutually exclusive with TPTRAN.

TPTRAN
: The service routine is in transaction mode. This setting is mutually exclusive with TPNOTRAN.

TPNOREPLY
: The program invoking the service routine is not expecting a reply. This setting is meaningful only when TPREQRSP is set. This setting is mutually exclusive with TPREPLY.

TPREPLY
: The program invoking the service routine is expecting a reply. This setting is meaningful only when TPREQRSP is set. This setting is mutually exclusive with TPNOREPLY.

TPSENDONLY
: The service is invoked such that it can send data across the connection and the program on the other end of the connection can only receive data. This setting is meaningful only when TPCONV is set. This setting is mutually exclusive with TPRECVONLY.

TPRECVONLY
: The service is invoked such that it can only receive data from the connection and the program on the other end of the connection can send data. This setting is meaningful only when TPCONV is set. This setting is mutually exclusive with TPSENDONLY.

RETURN VALUE

Upon successful completion, TPSVCSTART sets **TP-STATUS** to [TPOK]. If the size of the incoming message is larger than the size specified in **LEN** on input, TPTRUNCATE is set and only **LEN** bytes are moved into DATA-REC. The remaining bytes are discarded.

ERRORS

Under the following conditions, TPSVCSTART fails and sets **TP-STATUS** to one of the following values:

[TPEINVAL]
: Invalid arguments were given (for example, **LEN** is 0).

[TPEPROTO]
: TPSVCSTART was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPACALL, TPCALL, TPCONNECT, TPRETURN.

NAME

TPUNADVERTISE — unadvertise a service name

SYNOPSIS

```
01  SERVICE-NAME  PIC X(15).

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPUNADVERTISE" USING SERVICE-NAME TPSTATUS-REC.
```

DESCRIPTION

TPUNADVERTISE allows a server to unadvertise a service that it offers. By default, a server's services are advertised when it is booted and they are unadvertised when it is shutdown.

TPUNADVERTISE removes *SERVICE-NAME* as an advertised service for the server. *SERVICE-NAME* cannot be SPACES. Also, *SERVICE-NAME* should be 15 characters or fewer. Longer names are accepted and truncated to 15 characters. Care should be taken such that truncated names do not match other service names.

RETURN VALUE

Upon successful completion, TPUNADVERTISE sets **TP-STATUS** to [TPOK].

ERRORS

Under the following conditions, TPUNADVERTISE fails and sets **TP-STATUS** to one of the following values:

[TPEINVAL]
: *SERVICE-NAME* is SPACES.

[TPENOENT]
: *SERVICE-NAME* is not currently advertised by the server.

[TPEPROTO]
: TPUNADVERTISE was called in an improper context.

[TPESYSTEM]
: A communication resource manager system error has occurred. The exact nature of the error is determined in a product-specific manner.

[TPEOS]
: An operating system error has occurred. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TPADVERTISE.

Chapter 8

State Tables

This chapter contains state tables that show legal calling sequences for the XATMI routines.

Note: Lower-case function names represent both the C and COBOL versions of the function except where noted.

8.1 Interface Functions Allowed

Table 8-1 summarises the interface functions that may be invoked from the three types of XATMI AP entities: clients, request/response services and conversational services.

Name	Client	Request/Response Service	Conversational Service
tpalloc(C)	•	•	•
tpfree(C)	•	•	•
tprealloc(C)	•	•	•
tptypes(C)	•	•	•
tpservice(C)		•	•
TPSVCSTART(COBOL)		•	•
tpreturn		•	•
tpadvertise		•	•
tpunadvertise		•	•
tpacall	•	•	•
tpcall	•	•	•
tpcancel	•	•	•
tpgetrply	•	•	•
tpconnect	•	•	•
tpdiscon	•	•	•
tprecv	•	•	•
tpsend	•	•	•

• Function is allowed.

Table 8-1 Interface Functions Allowed by Type of Entity.

The following state tables represent calling sequences for valid uses of XATMI functions within each of the sets of functions in Table 3-1 on page 14 and Table 6-1 on page 56. Unless noted otherwise, incorrect use of any function does not affect the state of the caller. Additionally, the use of any function from one set does not affect the state of another set.

8.2 Typed Buffer Functions

Table 8-2 represents the state of the AP with respect to typed buffer functions. The states, that relate to a particular typed buffer, are as follows:

S_0 buffer not allocated
S_1 buffer allocated.

Name	S_0	S_1
tpalloc	S_1	
tpfree		S_0
tprealloc		S_1
tptypes		S_1

Table 8-2 State Table for Typed Buffer Functions

Note: Service routines may be invoked with a buffer already in state S_1.

8.3 Service Routine Functions

Table 8-3 represents the state of the AP with respect to functions for writing service routines. The states in this table are:

S_0 not in a service routine
S_1 in a service routine.

Name	S_0	S_1
tpservice(C)	S_1	
TPSVCSTART(COBOL)	S_1	
tpreturn		S_0

Table 8-3 State Table for Service Routine Functions

8.4 Advertising Functions

Table 8-4 represents the state of the AP with respect to functions for dynamically advertising service names. This table relates to the state of an individual service name; services may be advertised at server initialisation in an implementation-specific manner. The states in this table are:

S_0 service not advertised
S_1 service advertised.

Name	S_0	S_1
tpadvertise	S_1	
tpunadvertise		S_0

Table 8-4 State Table for Advertising Functions

8.5 Request/Response Service Functions

Table 8-5 represents the state of the AP with respect to functions for request/response services. This table relates to the state of an individual descriptor/handle. The states in this table are:

S_0 descriptor/handle is invalid
S_1 descriptor/handle is valid.

Service routines begin with one descriptor in state S_1.

Name	S_0	S_1
tpacall	S_1	
tpcancel		S_0
tpgetrply		S_0

Table 8-5 State Table for Request/Response Service Functions

8.6 Conversational Service Functions

Table 8-6 represents the state of the AP with respect to functions for conversational services. This table relates to the state of an individual descriptor/handle. The states in this table are:

S_0 no descriptor/handle
S_1 descriptor/handle in send only mode
S_2 descriptor/handle in receive only mode.

Service routines begin with one descriptor in state S_1 or state S_2.

In addition, the following notation is used:

S* SENDONLY flag used
S** RECVONLY flag not set
S+ TPEV-SENDONLY event received
R* RECVONLY flag set
R+ TPEV-SENDONLY event not received
A originator of conversation
B subordinate of conversation.

Name	S_0	S_1	S_2
tpconnect/S*	S_1		
tpconnect/R*	S_2		
tprecv/S+			S_1
tprecv/R+			S_2
tpsend/S**		S_1	
tpsend/R*		S_2	
tpdiscon/A		S_0	S_0
tpreturn/B		S_0	S_0

Table 8-6 State Table for Conversational Service Functions

Additionally, certain XATMI functions can affect the state of the caller's transaction, if any. Specifically, the following calls and conditions cause the CRM to mark the transaction rollback-only:

- *tpacall*( ), *tpcall*( ), *tpconnect*( ), *tpgetrply*( ), *tprecv*( ) or *tpsend*( ) returning [TPETIME]
- *tpcall*( ) or *tpgetrply*( ) returning [TPEOTYPE], [TPESVCERR] or [TPESVCFAIL] if these calls were issued on behalf of a global transaction
- *tprecv*( ) returning [TPEOTYPE] or [TPEEVENT] with one of TPEV_DISCONIMM, TPEV_SVCERR or TPEV_SVCFAIL if this call was issued on behalf of a global transaction.
- *tpsend*( ) returning [TPEEVENT] with one of TPEV_DISCONIMM, TPEV_SVCERR or TPEV_SVCFAIL if this call was issued on behalf of a global transaction.
- *tpdiscon*( ) issued on behalf of a global transaction.
- *tpreturn*( ) with TPFAIL set if the service was part of a global transaction.

Chapter 9

X/Open Specified Buffer and Record Types

This chapter contains detailed information on the C-language typed buffers and the COBOL typed records that all implementations of the XATMI interface must support.

9.1 C-language Buffer Types

X/Open specifies three buffer types for the C-language bindings of XATMI: X_OCTET, X_COMMON, and X_C_TYPE.

9.1.1 X_OCTET

The X_OCTET typed buffer is an array of bytes (characters) whose contents and handling are completely defined by the AP. This buffer type is also referred to as an octet array. As such, a length parameter always accompanies an X_OCTET typed buffer so that the CRM knows how many bytes in the character array to send. This length is provided as the input *len* parameter of a request and is retrieved in the output *len* parameter. Because data is transferred ''as is'' across heterogeneous machine boundaries, APs must agree on the interpretation of the buffer's contents. Any character in the array can be NULL. The X_OCTET typed buffer has no subtype.

The following illustrates how an AP could use this buffer type:

```
char *octet_ptr;
char *ptr1, *ptr2;

/* allocate space for 25 bytes */
octet_ptr = tpalloc("X_OCTET", NULL, 25);

ptr1 = octet_ptr;          /* point to start of typed buffer */
ptr2 = octet_ptr + 10;     /* point to eleventh byte in typed buffer */

strcpy(ptr1, "hello");     /* add first string to typed buffer */
strcpy(ptr2, "goodbye");   /* add second string to typed buffer */

/* send one character array containing two NULL-terminated strings */
tpacall("GREETSVC", octet_ptr, 25, TPNOREPLY);
```

9.1.2 X_COMMON

The X_COMMON typed buffer is a non-nested C structure whose elements can be any of the following three C data types: **short**, **long** or **char**. Bounded arrays of these elements are also allowed within an X_COMMON structure definition. The X_COMMON typed buffer can contain octet arrays whose contents and handling are completely defined by the AP; the XATMI CRM does no processing on them. Specific structure definitions for this buffer type must be qualified by an application-defined subtype. The manner in which X_COMMON subtypes are defined is implementation-specific. Once defined to an XATMI CRM, these subtypes (except for octet arrays) are transparently encoded and decoded on behalf of APs when they cross heterogeneous machine boundaries.

These data types have a mapping between their respective COBOL data types (**PIC S9(4) COMP-5**, **PIC S9(9) COMP-5** and **PIC X(1)**). A C-language AP using X_COMMON typed buffers can communicate with a COBOL AP using X_COMMON typed records so long as both APs recognise the same subtypes.

With respect to processing strings, because COBOL programs do not use a NULL character to terminate character strings, the most portable way for a program to communicate strings is via an accompanying length element that both the C and COBOL programs use to determine the number of significant bytes in a character array. This length can be implicitly known by both partners, or can be defined by mutual understanding as another field of the same buffer.

The following illustrates how an AP could use this buffer type. This example shows a C-language client calling DEPOSITSVC. In Section 9.2.2 on page 107, an example is given showing a COBOL language service routine using the same X_COMMON subtype.

Firstly, the AP defines an X_COMMON subtype. In this example, a subtype called **deposit** is used.

```
struct deposit {
        long    acct_no;
        short   amount;
        short   balance;
        char    status[128];
        short   status_len;
};
```

Next, this structure is processed by the CRM (in an implementation-specific manner) so that it can allocate memory for this subtype and process it appropriately at run time.

Finally, an AP could be written to use this subtype as follows:

```
struct deposit *dptr;
long len;

/* allocate space for a deposit structure */
dptr = tpalloc("X_COMMON", "deposit", 0);

/* populate buffer with data */
dptr->acct_no = 12345678;
dptr->amount  = 50;

/* call DEPOSITSVC, place reply in same buffer */
tpcall("DEPOSITSVC", (char *) dptr, 0, (char **) &dptr, &len,
       TPNOCHANGE);

/* call AP function to print balance and any status message returned */
bal_print(dptr->balance, dptr->status, dptr->status_len);
```

9.1.3 X_C_TYPE

The X_C_TYPE typed buffer is a non-nested C structure whose elements can be any of the following data types: **int**, **short**, **long**, **char**, **float**, **double**, character string and octet array. The first six types are basic C data types while the last two are both bounded arrays of characters. Strings are given credence because of the prevalence of NULL-terminated character arrays in C programs and libraries. This data type instructs the XATMI CRM to process a string element only up until a NULL character is reached. Octet arrays are also arrays of characters, any of which may be NULL. Like the X_OCTET buffer type, the contents and handling of octet arrays are completely defined by the AP; the XATMI CRM does no processing on them. All the other elements of an X_C_TYPE buffer type, as for the X_COMMON buffer type, are transparently encoded and decoded by the XATMI CRM, provided they are defined to it. Note that the X_COMMON typed buffer is a strict subset of X_C_TYPE, that adds only **int**s, **float**s, **double**s, character strings and octet arrays.

Bounded arrays of these eight data types are allowed within an X_C_TYPE structure definition. For strings and octet arrays, this implies that a structure could contain arrays of strings and arrays of octet arrays. Specific structure definitions for this buffer type must be qualified by an application-defined subtype. The manner in which X_C_TYPE subtypes are defined is implementation-specific. Once defined to an XATMI CRM, these subtypes (except for octet arrays) are transparently encoded and decoded on behalf of APs when they cross heterogeneous machine boundaries.

The following illustrates how an AP could use this buffer type.

First, the AP defines an X_C_TYPE subtype. In this example, a subtype called "acct_info" is used.

```
struct acct_info {
        long    acct_no;
        char    name[50];       /* NULL terminated string */
        char    address[100];   /* NULL terminated string */
        float   balances[2];    /* both checking and savings balances */
};
```

Next, this structure is processed by the CRM (in an implementation-specific manner) so that it can allocate memory for this subtype and process it appropriately at run time.

Finally, an AP could be written to use this subtype as follows:

```
struct acct_info *aptr;
long len;

/* allocate space for a deposit structure */
aptr = tpalloc("X_C_TYPE", "acct_info", 0);

/* populate buffer with data */
aptr->acct_no = 12345678;

/* call INQUIRY, place reply in same buffer */
tpcall("INQUIRY", (char *) aptr, 0, (char **) &aptr, &len, TPNOCHANGE);

/* call AP function to print account information returned */
acct_print(aptr->acct_no, aptr->name, aptr->address,
           aptr->balance[0], aptr->balance[1]);
```

9.2 COBOL Language Buffer Types

X/Open specifies two buffer types for the COBOL language bindings of XATMI: X_OCTET and X_COMMON.

9.2.1 X_OCTET

The X_OCTET record type is a set of contiguous bytes (characters) whose contents and handling are completely defined by the AP. This record type is also referred to as an octet array. As such, a length parameter always accompanies an X_OCTET record type so that the CRM knows how many bytes to send. This length is to be provided in the **LEN** field of the input *TPTYPE-REC* record of a request, and is retrieved in the **LEN** field of the output *TPTYPE-REC* record. Because data is transferred ''as is'' across heterogeneous machine boundaries, APs must agree on the interpretation of the record's contents. Any character in the record can be a LOW-VALUE. The X_OCTET record type has no subtype.

The following illustrates how an AP could use this buffer type.

```
 01 OCTET-REC.
    05 HELLO-FIELD PIC X(5) VALUE IS "Hello".
    05 FILLER      PIC X(1) VALUE IS LOW-VALUE.
    05 GOODBYE-FIELD PIC X(7) VALUE IS "Goodbye".
    05 FILLER      PIC X(1) VALUE IS LOW-VALUE.
*
* Set up TPTYPE-REC
*
 MOVE "X_OCTET" TO REC-TYPE.
 MOVE SPACES TO SUB-TYPE.
 MOVE LENGTH OF OCTET-REC TO LEN.
*
* Set up TPSVCDEF-REC
*
 MOVE "GREETSVC" TO SERVICE-NAME.
*
* Send octet record to GREETSVC.
*
 CALL "TPACALL" USING
         TPSVCDEF-REC TPTYPE-REC OCTET-REC TPSTATUS-REC.
```

9.2.2 X_COMMON

The X_COMMON record type is a non-nested COBOL record whose elements can be any of the following three COBOL data types: **PIC S9(4) COMP-5**, **PIC S9(9) COMP-5** and **PIC X(1)**. Multiple occurrences of these elements are also allowed within an X_COMMON record definition (via the use of an **OCCURS** clause, or a **PIC X(*n*)** definition for multiple **PIC X(1)**). The X_COMMON record type can contain octet arrays whose contents and handling are completely defined by the AP; the XATMI CRM does no processing on them. Specific record definitions for this record type must be qualified by an application-defined subtype. The manner in which X_COMMON subtypes are defined is implementation-specific. Once defined to an XATMI CRM, these subtypes (except for octet arrays) are transparently encoded and decoded on behalf of APs when they cross heterogeneous machine boundaries.

These data types have a one-to-one mapping between their respective C data types (**short**, **long** and **char**). A COBOL-language AP using X_COMMON record types can communicate with a C-language AP using X_COMMON typed buffers so long as both APs recognise the same subtypes.

With respect to processing strings, because COBOL programs do not use a NULL character to terminate character strings (as do C programs), the most portable way for a program to communicate strings is via an accompanying length element that both the C and COBOL programs use to determine the number of significant bytes in a character array.

The following illustrates how an AP could use this record type. This example is a continuation of that shown in Section 9.1.2 on page 104 where a C-language client calls DEPOSITSVC. Shown below is the COBOL version of DEPOSITSVC using the same X_COMMON subtype.

First, the AP defines an X_COMMON subtype. In this example, the subtype is the COBOL equivalent to that shown section 8.1.2 and it is located in a COPY file called DEPOSIT.

```
05 ACCT-NO    PIC S9(9) COMP-5.
05 AMOUNT     PIC S9(4) COMP-5.
05 BALANCE    PIC S9(4) COMP-5.
05 STATUS     PIC X(128).
05 STATUS-LEN PIC S9(4) COMP-5.
```

Next, this record is processed by the CRM (in an implementation-specific manner) so that it can process it appropriately at run time.

Finally, an AP could be written to use this subtype as follows:

```
 01  DEPOSIT-REC.
 COPY DEPOSIT.
*
 MOVE LENGTH OF DEPOSIT-REC TO LEN.
*
 CALL "TPSVCSTART" USING
        TPSVCDEF-REC TPTYPE-REC DEPOSIT-REC TPSTATUS-REC.
*
* PROCESS-DEPOSIT accesses DBMS and returns BALANCE and STATUS.
*
 CALL "PROCESS-DEPOSIT" USING DEPOSIT-REC.

 IF NO ERRORS
        SET TPSUCCESS TO TRUE
 ELSE
        SET TPFAIL TO TRUE
*
 COPY TPRETURN REPLACING DATA-REC BY DEPOSIT-REC.
```

X/Open CAE Specification

Part 2:

XATMI Application Service Element (ASE)

X/Open Company Ltd.

Chapter 10

XATMI Communication Model

This chapter describes the mapping of the communication model used by the X/Open XATMI Interface to the OSI TP Communication Model. This mapping is abstracted through the definition of the XATMI Application Service Element (XATMI-ASE) which follows the OSI ASE definition nomenclature.

10.1 XATMI-ASE Communication Model

The XATMI-ASE defines how the primitives in the XATMI Interface are mapped to the OSI TP protocol. This protocol is *connection-oriented* and assumes that the cooperating applications communicate via dialogues.

An OSI TP dialogue is established in the XATMI-ASE Model when a client requests either a request/response or a conversational service. This dialogue may be established within or outside the current global transaction. A dialogue is accepted or rejected by the remote server (or application) and a service is invoked as the result of the incoming dialogue request within or outside the global transaction (as requested by the client).

A new dialogue is allocated with each service request and the lifetime of the dialogue is subject to the duration of the transaction (if the dialogue is within a transaction) or to the duration of the service (if the dialogue is not within a transaction).

The communication model used by Conversational Services maps to polarised dialogues, that is, only one side (the client or the service) may send data at a time.

For simplicity, the definitions of the XATMI-ASE mappings do not show a service acting as a client. The mappings for that case, however, can be easily inferred from the defined client mappings.

10.2 OSI TP Profiles

The XATMI-ASE maps to OSI TP profiles ATP11, ATP21, and ATP31 (see ISO/IEC ISP 12061). The following table summarises the OSI TP functional units required by each one of these profiles (a • symbol indicates that the specified functional unit is required for that profile):

Functional Units	ATP11	ATP21	ATP31
Dialogue	•	•	•
Polarized Control	•	•	•
Shared Control			
Commit		•	•
Unchained transactions		•	
Chained transactions			•
Handshake	•	•	•
Recovery		•	•

Table 10-1 Required OSI TP Functional Units

The choice between these profiles is automatically supported by the XATMI-ASE. The ATP11 profile is used when a service request is issued outside a global transaction. The ATP21 or the ATP31 profile is selected when the XATMI-ASE is used within a global transaction (that is, the selection is provider-dependent but profile ATP21 is the default).

10.3 Structure of the XATMI-ASE

The basic structure of the XATMI-ASE is derived from the OSI Application Layer Structure, described in ISO/IEC 9545, and ISO/IEC Distributed Transaction Processing, described in the referenced OSI TP standards.

The following figure illustrates the functional architecture of the XATMI-ASE from the OSI point of view and is not meant to imply any particular software architecture.

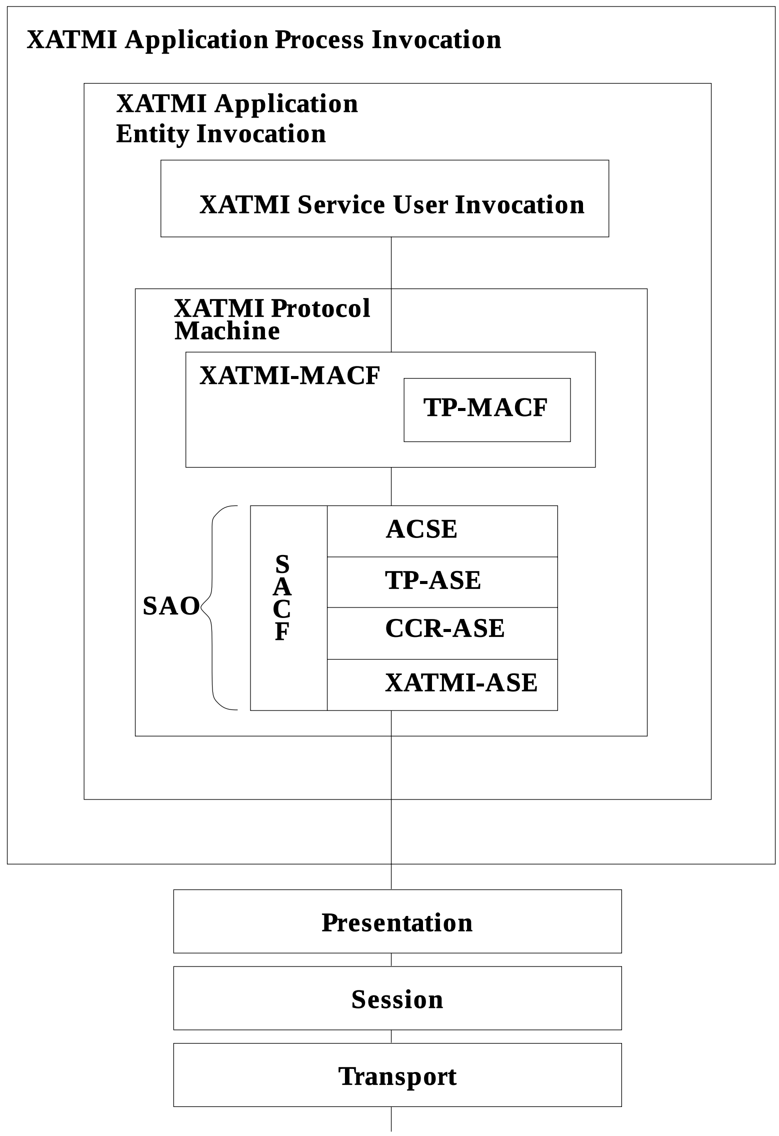


Figure 10-1 OSI View of XATMI-ASE Functional Architecture

The main elements of this architecture are as follows:

XATMI Application Process Invocation (XATMI-API)
An Application Process is the OSI abstraction of an application (including all its communication functions). An Application Process Invocation represents a particular instantiation of an application which includes all of the elements illustrated in the figure above. Therefore, an XATMI-API represents a particular instantiation of an application using the XATMI Interface.

XATMI Service User Invocation (XATMI-SUI)
An XATMI-SUI represents the actual *user* of the services provided by the XATMI Protocol Machine (XATMI-PM). Hence, an XATMI-SUI represents a particular instantiation of the implementation of the XATMI Interface (that is, the XATMI Provider).

XATMI Application Entity Invocation (XATMI-AEI)
An XATMI Application Entity (XATMI-AE) represents the component of an XATMI-AP that provides the communication services necessary for interworking in the XATMI OSI environment. An XATMI-AEI refers to the specific use of the services of the XATMI-AE by a particular XATMI-API.

XATMI Protocol Machine (XATMI-PM)
An XATMI-PM represents the component of an XATMI-AEI that coordinates the sequencing and the mapping of the XATMI communication services to the OSI TP protocol (see Chapter 13). The XATMI-PM contains a complete OSI TP Protocol Machine (TPPM). The two primary components of the XATMI-PM are the XATMI Multiple Association Control Function (XATMI-MACF) and the Single Association Object (SAO). An XATMI-PM can have multiple SAOs that are coordinated by the XATMI-MACF.

XATMI Multiple Association Control Function (XATMI-MACF)
The XATMI-MACF ensures the proper sequencing of the TP protocol and the XATMI Application Protocol Data Units (APDUs) flowing across TP dialogues (that is, a dialogue maps to an association). The XATMI-MACF is a supplement to the TP-MACF.

Single Association Object (SAO)
An SAO represents the context used within a particular association. Each SAO contains a Single Association Control Function (SACF) and a set of Application Service Elements (ASEs) that supports the specific communication required by an XATMI-SUI.

Single Association Control Function (SACF)
The SACF abstracts the control required for coordinating the use of a single association by multiple ASEs and their use of the Presentation Service. In the XATMI OSI Communication Model, the SACF coordinates the following ASEs:

- the OSI Association Control Service Element (ACSE)
- the OSI TP Application Service Element (TP-ASE)
- the OSI Commitment, Concurrency, and Recovery Application Service Element (CCR-ASE)
- the XATMI Application Service Element (XATMI-ASE).

XATMI Application Service Element (XATMI-ASE)
The XATMI-ASE provides the service primitives that handle the data transfer for the request/response and Conversational Service requests (see Chapter 12). Also, the XATMI-ASE provides the necessary protocol encoding and decoding services for the XATMI-PM (see Chapter 12 and Chapter 14).

10.4 OSI TP Naming Model

OSI TP communication requires the following naming structure:

Application Process Title (APT)
An APT identifies a particular application in the OSI network. An APT must be registered with a registration authority.

Application Entity Qualifier (AEQ)
An AEQ identifies the particular communication functions that are used with an association.

Application Entity Title (AET)
An AET identifies an XATMI-AEI within the OSI environment. An AET is always formed by the combination of the APT and the AEQ.

Application Context Name (ACN)
An ACN identifies rules for the communication between two XATMI-AEIs. An ACN is an Object Identifier that must be registered with a registration authority. In particular, X/Open must register an ACN to identify the XATMI OSI context (see Chapter 11).

Transaction Processing Service User Title (TPSUT)
A TPSUT identifies a remote program (or TP Service User — TPSU) within a particular XATMI-API.

Abstract Syntax Name (ASN)
An ASN defines the structure used for the exchange of information between peer ASEs. An ASN is an Object Identifier that must be registered with a registration authority. In particular, X/Open must register an ASN to identify the APDUs used by the XATMI-ASE (see Chapter 14).

The mapping of these names from the XATMI Interface and the XATMI OSI Communication Model to the OSI TP Model is defined in Chapter 13.

10.5 XATMI-PM and the X/Open DTP Model

The following figure shows the relationship between the XATMI-PM and the X/Open Distributed Transaction Processing (DTP) Model. In particular, the figure shows where the different X/Open transaction interfaces relate to an implementation of the XATMI-PM.

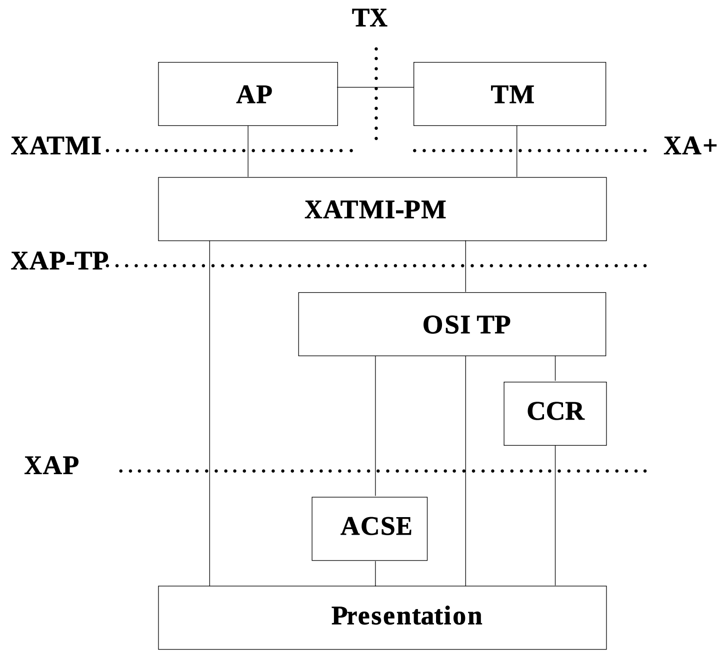


Figure 10-2 Relationship Between XATMI-PM and DTP Model

Chapter 11

XATMI Application Context Definition

This chapter contains the full application context for use with the XATMI-ASE. Its purpose is to provide an application context that supports TP applications using the XATMI Interface. Bilateral agreements are required between the XATMI-SUIs (or TPSUIs) with respect to the structure of the typed buffers supported by the cooperating applications.

This chapter covers the application context identifier, the component ASEs, SACF rules and MACF rules. There are no other functions and no persistent application context rules.

11.1 Application Context Identifier

X/Open must register an Object Identifier that identifies this application context, for example:

```
{iso(1) national-member-body(2) bsi(826) disc(0) xopen(1050)
        xatmi(4) application-context(2) atp11-21-31(1)}
```

11.2 Component ASEs

The following ASEs are contained in this application context:

- ACSE
- CCR (optional, included only if the TP commit functional unit is required)
- TP-ASE
- XATMI-ASE.

ACSE

References	ISO / IEC 8649 and 8650
Version	Version 1
Description	ACSE is used to establish and terminate associations. The ACSE functions are not exercised directly by the XATMI-PM but are exercised by association management facilities within the TP service provider.

CCR

References	ISO / IEC 9804 and 9805
Version	Version 2
Description	CCR is used in support of the commitment, rollback, and recovery functions. TP makes use of the CCR ASE services. The XATMI-PM does not make a direct use of CCR functions.

TP-ASE

References	The OSI TP standards
Version	Version 1
Description	TP-ASE is used to provide functions that are specific to TP. The XATMI-PM uses the TP services.

XATMI-ASE

References	This document
Version	Version 1
Description	The XATMI-ASE defines the characteristics of the transfer of structured data between cooperating TPSUIs that use the X/Open XATMI Interface specification. The Abstract Syntax of the XATMI-ASE APDUs is defined in Chapter 14. The Transfer Syntax is defined by the Basic Encoding Rules for ASN.1 (see the referenced BER standard).

11.3 SACF Rules

The SACF rules defined for the XATMI-ASE are in addition to the rules defined for the TP-ASE. Chapter 13 defines these rules.

11.3.1 Sequencing Rules

Chapter 13 defines the correct ordering for the mapping of the XATMI-ASE services to the services provided by the TP-ASE.

11.3.2 Concatenation Rules

There are no concatenation rules beyond those specified in the base standard (the OSI TP Protocol standard).

11.3.3 Mapping Rules

The following XATMI-ASE APDUs are mapped according to the concatenation rules specified in the OSI TP Protocol standard for the abstract service TP-DATA:

- XATMI-CALL-RI
- XATMI-REPLY-RI
- XATMI-CONNECT-RI
- XATMI-DATA-RI.

The XATMI-FAILURE-RI APDU is mapped to the User-Data parameter of the TP-U-ABORT service.

11.3.4 Transaction States

There are no additional transaction state transitions beyond those specified in the base standard (the OSI TP Protocol standard) for the TP-DATA generic service.

11.4 MACF Rules

The MACF rules defined for the XATMI-ASE are in addition to the rules defined by the TP-ASE. The XATMI-ASE allows:

- multiple asynchronous service requests, each optionally issued as part of a transaction.
- multiple conversational service requests, each optionally issued as part of a transaction
- two-phase commitment integrated with the X/Open Distributed Transaction Processing Model.

11.4.1 Sequencing Rules

The sequencing rules are defined in Chapter 13. These rules define how the XATMI-ASE services and APDUs are mapped to the TP-ASE. In particular, Section 13.20.1 on page 178 defines how the XATMI-PREPARE request service is mapped to all TP dialogues associated with the transaction.

11.4.2 Concatenation Rules

There are no MACF concatenation rules beyond those specified in the base standard (the OSI TP Protocol standard).

11.4.3 Mapping Rules

XATMI-MACF services are mapped one to one to XATMI-ASE services, except for the transaction services that are mapped directly to the corresponding TP services (see Chapter 12 and Chapter 13). From these transaction services, XATMI-PREPARE request and XATMI-READY indication require special handling by the XATMI-MACF as described in sections Section 13.20.1 on page 178 and Section 13.20.5 on page 180.

XATMI-ASE services are mapped directly to TP-ASE services as specified in Chapter 13.

Chapter 12

XATMI-ASE Service Definition

This chapter presents the XATMI-ASE services, mapping from the XATMI interface, sequencing rules and state table.

12.1 Nomenclature

The definitions of the XATMI-ASE service parameters use the following convention to describe the presence of each parameter:

blank: not applicable
M: presence is mandatory
U: presence is a user option
O: presence is a provider option
C: presence is conditional.

In addition, the notation (=) indicates that a parameter value is semantically equivalent to the parameter in the service primitive to its immediate left in the table.

Finally, this document uses the XATMI C language bindings to present the different mappings between the XATMI Interface and the XATMI-ASE definition. It is assumed that the XATMI COBOL language mappings are built upon the XATMI C-language mappings.

12.2 Summary of Service Primitives

The following table presents a summary of the XATMI-ASE service primitives used by an XATMI-SUI in the request/response and conversational communication models. Note that an XATMI-SUI takes the *client role* when it describes the behaviour of both a client application and a server application (that is, an application service) acting as a client of another server. An XATMI-SUI takes the *server role* when it describes the behaviour of a server application.

Key:

R/R request/response

Con conversational

• indicates the use of this XATMI-ASE service from that particular XATMI communication model.

Service Name	Client Role	Server Role	R/R	Con	Service Description	Protocol Description
XATMI-CALL	req	ind	•		Section 12.4.1 on page 124.	Section 13.6 on page 153 and Section 13.13 on page 164.
XATMI-REPLY	ind	req	•	•	Section 12.4.2 on page 126	Section 13.8 on page 159 and Section 13.15 on page 169.
XATMI-FAILURE	ind	req	•	•	Section 12.4.3 on page 127	Section 13.9 on page 160 and Section 13.16 on page 170.
XATMI-CANCEL	req	ind	•		Section 12.4.4 on page 129.	Section 13.10 on page 161 and Section 13.17 on page 175.
XATMI-CONNECT	req	ind		•	Section 12.4.5 on page 130	Section 13.7 on page 157 and Section 13.14 on page 167.
XATMI-DISCON	req	ind		•	Section 12.4.6 on page 132.	Section 13.12 on page 163 and Section 13.18 on page 176.
XATMI-DATA	req/ind	req/ind		•	Section 12.4.7 on page 133.	Section 13.11 on page 162 and Section 13.19 on page 177.
XATMI-PREPARE	req	ind	•	•	Section 12.4.8 on page 135.	Section 13.20.1 on page 178 and Section 13.20.5 on page 180.
XATMI-READY	ind		•	•	Section 12.4.9 on page 136.	Section 13.20.6 on page 180.
XATMI-COMMIT	req/ind	req/ind	•	•	Section 12.4.10 on page 137.	Section 13.20.2 on page 178 and Section 13.20.7 on page 181.
XATMI-DONE	req	req	•	•	Section 12.4.11 on page 138.	Section 13.20.3 on page 179.
XATMI-COMPLETE	ind	ind	•	•	Section 12.4.12 on page 139.	Section 13.20.9 on page 182.
XATMI-ROLLBACK	req/ind	ind	•	•	Section 12.4.13 on page 140.	Section 13.20.4 on page 180. and Section 13.20.8 on page 181.
XATMI-HEURISTIC	ind		•	•	Section 12.4.14 on page 141.	Section 13.20.10 on page 182.

Table 12-1 XATMI-ASE Service Primitives

The XATMI-ASE transaction service primitives enable an XATMI-SUI to commit or participate in the commitment of a transaction, to roll back a transaction and to receive the result of the completion of the commitment or rollback of a transaction. These services are *global* services, that is they affect all of the OSI TP dialogues controlled by the XATMI-PM on behalf of a transaction.

12.3 Mapping from the XATMI Interface

The following table summarises the complete set of XATMI-ASE services used by the relevant XATMI interface primitives.

XATMI Interface Primitive	Client Role	Server Role	XATMI-ASE Services
tpcall( )	•		XATMI-CALL req (XATMI-REPLY ind or XATMI-FAILURE ind)
tpacall( )	•		XATMI-CALL req
tpgetrply( )	•		XATMI-REPLY ind or XATMI-FAILURE ind
tpcancel( )	•		XATMI-CANCEL req
tpservice( )		•	XATMI-CALL ind or XATMI-CONNECT ind
tpreturn( )		•	XATMI-REPLY req or XATMI-FAILURE req
tpconnect( )	•		XATMI-CONNECT req
tpdiscon( )	•		XATMI-DISCON req
tpsend( )	•	•	XATMI-DATA req (XATMI-REPLY ind or XATMI-DISCON ind or XATMI-FAILURE ind)
tprecv( )	•	•	XATMI-DATA ind and (XATMI-REPLY ind or XATMI-DISCON ind or XATMI-FAILURE ind)

Table 12-2 XATMI-ASE Services Used by XATMI Interface Primitives

The following table summarises the complete set of XATMI-ASE transaction services used by the relevant X/Open TX interface primitives. These primitives are assumed to be used by an XATMI-SUI taking the *client role*.

TX Interface Primitive	XATMI-ASE Services
tx_commit( )	XATMI-PREPARE req XATMI-READY ind XATMI-COMMIT req XATMI-COMMIT ind XATMI-DONE req XATMI-COMPLETE ind
tx_rollback( )	XATMI-ROLLBACK req XATMI-DONE req XATMI-COMPLETE ind

Table 12-3 XATMI-ASE Services Used by TX Interface Primitives

12.4 XATMI-ASE Services

For each XATMI-ASE service, the request and indication are defined under the headings **Parameters** and **Usage**.

12.4.1 XATMI-CALL request and indication

Parameters

Parameter Name	**req**	**ind**
Service-Name	M	M
User-Data	U	U(=)
Begin-Transaction	M	M(=)
No-Reply-Option	M	M(=)

Service-Name
This parameter specifies a symbolic name pointing to local configuration information that is used by the XATMI-PM to extract the parameters necessary to establish an OSI TP dialogue with the remote server. The server XATMI-PM retrieves this name from the XATMI-CALL-RI APDU (see Chapter 13 and Chapter 14).

User-Data
This parameter specifies a typed buffer. The XATMI-PM uses the buffer's type to encode or decode the user data according to the rules specified in Chapter 14.

Begin-Transaction
This parameter specifies whether the XATMI-PM should issue the service request as part of the caller's global transaction, if any exists. It must take one of the following values: **True**, when the application service request should be part of the caller's transaction; **False**, when the application service request should not be part of the caller's transaction.

No-Reply-Option
This parameter is used to indicate to the XATMI-ASE whether a reply should be issued in response to the service request. It must take one of the following values: **True**, when a reply should not be issued; **False**, when a reply should be issued.

Usage

An XATMI-CALL request is mapped from *tpcall*( ) or *tpacall*( ) and is issued by a client to an XATMI-PM to request a remote service.

An XATMI-CALL indication is mapped to the *tpservice*( ) abstraction and is issued by the XATMI-PM to the server to invoke a request/response service.

Once an XATMI-CALL request has been issued by the client XATMI-SUI, one of the following events can occur:

- issue an XATMI-CANCEL request
- issue an XATMI-ROLLBACK request
- receive an XATMI-REPLY indication
- receive an XATMI-FAILURE indication.

Once an XATMI-CALL indication has been received by the server XATMI-SUI, one of the following events can occur:

- issue an XATMI-REPLY request
- issue an XATMI-FAILURE request
- receive an XATMI-CANCEL indication
- receive an XATMI-ROLLBACK indication.

12.4.2 XATMI-REPLY request and indication

Parameters

Parameter Name	**req**	**ind**
User-Code	M	M(=)
User-Data	U	U(=)

User-Code
: This parameter specifies the return code defined by the application using the XATMI interface. This parameter is encoded by the XATMI-PM according to the rules specified in Chapter 14.

User-Data
: This parameter specifies a typed buffer. The XATMI-PM uses the buffer's type to encode or decode the user data according to the rules specified in Chapter 14.

Usage

An XATMI-REPLY request is issued by a server to the XATMI-PM to return the reply from the service. It is mapped from *tpreturn*( ) with *rval* set to TPSUCCESS.

An XATMI-REPLY indication is issued by the XATMI-PM to the client to return the reply from the remote service. It is mapped to *tpcall*( ), to *tpgetrply*( ) or to *tprecv*( ).

Once an XATMI-REPLY request has been issued by a server XATMI-SUI, one of the following events can occur:

- receive an XATMI-ROLLBACK indication
- receive an XATMI-PREPARE indication.

Once an XATMI-REPLY indication has been received by a client XATMI-SUI, one of the following events can occur:

- issue an XATMI-PREPARE request
- issue an XATMI-ROLLBACK request
- receive an XATMI-ROLLBACK indication.

12.4.3 XATMI-FAILURE request and indication

Parameters

Parameter Name	req	ind
Diagnostic	M	M
User-Code	C	C(=)
User-Data	C	C(=)

Diagnostic

This parameter specifies the failure type. It may be set by the server, the server XATMI-PM, or the client XATMI-PM. The diagnostic is always reported on the indication. The following table summarises the Diagnostic valid values (see Section 13.9 on page 160 and Section 13.16 on page 170 for the corresponding mapping to the XATMI Interface):

Diagnostic Value	req	ind
Application-Service-Failure	•	•
Recipient-XATMI-SU-Failure	•	•
Rejected-XATMI-Provider		•
Permanent-Failure		•
Transient-Failure		•
Protocol-Error		•
Recipient-Unknown		•
Recipient-TPSU-title-unknown		•
Recipient-TPSU-title-required		•
TPSU-not-available(permanent)		•
TPSU-not-available(transient)		•
Functional-Unit-combination-not-supported		•
Reason-not-specified		•

User-Code

This parameter specifies the return code defined by the application using the XATMI interface and is present only if the Diagnostic parameter is set to **Application-Service-Failure**. This parameter is encoded by the XATMI-PM according to the rules specified in Chapter 14.

User-Data

This parameter specifies a typed buffer and may be present only if the Diagnostic parameter is set to **Application-Service-Failure**. The XATMI-PM uses the buffer's type to encode or decode the user data according to the rules specified in Chapter 14.

Usage

An XATMI-FAILURE request is issued by a server to the XATMI-PM to return a failure from the service. Normally, an XATMI-FAILURE is mapped from a *tpreturn*( ) with *rval* set to TPFAIL. The XATMI Provider, however, may also issue an XATMI-FAILURE if an error condition is found after the application service returns (see XATMI Interface for details).

An XATMI-FAILURE indication is issued by the XATMI-PM to the client to return the failure from the remote service. It is mapped to *tpcall*( ), *tpgetrply*( ), *tpsend*( ) or *tprecv*( ).

Once an XATMI-FAILURE request has been issued by a server XATMI-SUI, only the following event can occur:

- issue an XATMI-DONE request.

Once an XATMI-FAILURE indication has been received by a client XATMI-SUI, only the following event can occur:

- issue an XATMI-DONE request.

12.4.4 XATMI-CANCEL request and indication

Parameters

None.

Usage

An XATMI-CANCEL request is issued by a client to the XATMI-PM to cancel a pending service reply. It applies only to request/response application services. It is mapped from *tpcancel*( ) (note that this function cannot be called when the client is within a transaction, see the manual page for *tpcancel*( ) on page 33).

An XATMI-CANCEL indication is issued by the XATMI-PM to the server to indicate the cancel from the client. This service is not mapped to the XATMI server interface.

Once an XATMI-CANCEL indication has been issued by a client XATMI-SUI or received by a server XATMI-SUI, no more events relating to the corresponding application service request can occur.

12.4.5 XATMI-CONNECT request and indication

Parameters

Parameter Name	**req**	**ind**
Service-Name	M	M
User-Data	U	U
Begin-Transaction	M	M(=)
Grant-Control	M	M(=)

Service-Name

This parameter specifies a symbolic name pointing to local configuration information that is used by the XATMI-PM to extract the parameters necessary to establish an OSI TP dialogue with the remote server. The server XATMI-PM retrieves this name from the XATMI-CONNECT-RI APDU (see Chapter 13 and Chapter 14).

User-Data

This parameter specifies a typed buffer. The XATMI-PM uses the buffer's type to encode or decode the user data according to the rules specified in Chapter 14.

Begin-Transaction

This parameter specifies whether the XATMI-PM should issue the service request as part of the caller's global transaction, if any exists. It must take one of the following values: **True**, when the application service request should be part of the caller's transaction; **False**, when the application service request should not be part of the caller's transaction.

Grant-Control

This parameter is used to indicate whether the client wishes to retain the control of the connection. It must take one of the following values: **False**, when the client retains the control of the connection; **True**, when the client grants control of the connection to the service.

Usage

An XATMI-CONNECT request is mapped from *tpconnect*( ) and is issued by a client to an XATMI-PM to request a connection with a remote conversational service.

An XATMI-CONNECT indication is mapped to the *tpservice*( ) abstraction and is issued by the XATMI-PM to the server to invoke a conversational service.

Connections with a conversational service are established in a such a way that only one side may send data at a time.

If the client retains control of the connection, one of the following events may occur:

- At the client side:
  - — issue an XATMI-DATA request
  - — issue an XATMI-DISCON request
  - — issue an XATMI-ROLLBACK request
  - — receive an XATMI-FAILURE indication
  - — receive an XATMI-REPLY indication.

- At the server side:
  — receive an XATMI-DATA indication
  — receive an XATMI-DISCON indication
  — receive an XATMI-ROLLBACK indication
  — issue an XATMI-REPLY request
  — issue an XATMI-FAILURE request.

If the client does not retain control of the connection, one of the following events can occur:

- At the client side:
  — issue an XATMI-DISCON request
  — issue an XATMI-ROLLBACK request
  — receive an XATMI-REPLY indication
  — receive an XATMI-FAILURE indication
  — receive an XATMI-DATA indication.
- At the server side
  — issue an XATMI-REPLY request
  — issue an XATMI-FAILURE request
  — issue an XATMI-DATA request
  — receive an XATMI-ROLLBACK indication
  — receive an XATMI-DISCON indication.

12.4.6 XATMI-DISCON request and indication

Parameters

None.

Usage

An XATMI-DISCON request is issued by the client to the XATMI-PM to end a conversation with a conversational service. It is mapped from *tpdiscon*( ).

An XATMI-DISCON indication is received by the server from the XATMI-PM. It may be generated because of a communication failure or because the client issued *tpdiscon*( ), and it indicates the end of the conversation with the client. If the conversation was established within a transaction, an XATMI-DISCON indication also implies that the transaction is rolling back.

An XATMI-DISCON indication is mapped to the TPEV_DISCONIMM event returned in the *revent* variable of *tpsend*( ) or *tprecv*( ).

Once an XATMI-DISCON request has been issued by the client, only the following event can occur for that transaction:

- issue an XATMI-ROLLBACK request.

Once an XATMI-DISCON request has been received by the server, only the following event can occur for that transaction:

- issue an XATMI-DONE request.

12.4.7 XATMI-DATA request and indication

Parameters

Parameter Name	req	ind
User-Data	U	U(=)
Grant-Control	M	M(=)

User-Data

This parameter specifies a typed buffer. The XATMI-PM uses the buffer's type to encode or decode the user data according to the rules specified in Chapter 14.

Grant-Control

This parameter is used to indicate whether the client (or the service) relinquishes the control of the connection. It must take one of the following values: **False**, when the XATMI-SUI retains or does not obtain control of the connection; **True**, when the XATMI-SUI grants or obtains control of the connection.

Usage

An XATMI-DATA request is mapped from *tpsend*( ) and is issued by a client or a server to send a typed buffer over a conversational connection. The issuer must have control of the connection.

An XATMI-DATA indication is mapped to *tprecv*( ) and is issued by the XATMI-PM to indicate that data has been received and is now available.

If the client retains control of the connection, one of the following events may occur:

- At the client side:
  - — issue an XATMI-DATA request
  - — issue an XATMI-DISCON request
  - — issue an XATMI-ROLLBACK request
  - — receive an XATMI-FAILURE indication
  - — receive an XATMI-REPLY indication.
- At the server side:
  - — receive an XATMI-DATA indication
  - — receive an XATMI-DISCON indication
  - — receive an XATMI-ROLLBACK indication
  - — issue an XATMI-REPLY request
  - — issue an XATMI-FAILURE request.

If the server receives control of the connection, one of the following events can occur:

- At the client side:
  - — issue an XATMI-DISCON request
  - — issue an XATMI-ROLLBACK request
  - — receive an XATMI-REPLY indication
  - — receive an XATMI-FAILURE indication
  - — receive an XATMI-DATA indication.
- At the server side:
  - — issue an XATMI-REPLY request
  - — issue an XATMI-FAILURE request
  - — issue an XATMI-DATA request
  - — receive an XATMI-ROLLBACK indication
  - — receive an XATMI-DISCON indication.

12.4.8 XATMI-PREPARE request and indication

Parameters

None.

Usage

An XATMI-PREPARE request is issued by the client to the XATMI-PM to start the first phase of commitment of the current transaction. It is mapped from *tx_commit*( ). An XATMI-PREPARE request is a *global* service and it applies to all instances of the XATMI-PM participating in that transaction.

An XATMI-PREPARE indication is received by the server from the XATMI-PM to indicate that all local recoverable resources must be placed into the Ready state. An XATMI-PREPARE indication applies to a particular instance of the server's XATMI-PM. There is no mapping of this indication to the XATMI interface.

Once an XATMI-PREPARE indication has been received by the server, one of the following events can occur for that transaction:

- issue an XATMI-COMMIT request to indicate that all local resources are ready
- issue an XATMI-ROLLBACK request if local resources cannot be placed into the Ready state
- receive an XATMI-ROLLBACK indication.

Once an XATMI-PREPARE request has been issued by the client, one of the following events can occur for that transaction:

- receive an XATMI-READY indication when all subordinate transaction branches are known to be in the READY state
- receive an XATMI-ROLLBACK indication.

12.4.9 XATMI-READY indication

Parameters

None.

Usage

An XATMI-READY indication is a *global* transaction service that is received by the client from the the XATMI-PM to indicate that all participants in the transaction are in the Ready state.

There is no mapping of this indication to the XATMI interface.

Once an XATMI-READY indication has been received by the client, one of the following events can occur for that transaction:

- issue an XATMI-COMMIT request to start the second phase of the commitment
- issue an XATMI-ROLLBACK request if local resources cannot be placed into the Ready state
- receive an XATMI-ROLLBACK indication.

12.4.10 XATMI-COMMIT request and indication

Parameters

None.

Usage

An XATMI-COMMIT request is a *global* transaction service that is issued by a client to the XATMI-PM when all local resources are in the Ready state and the second phase of commitment is to be started.

An XATMI-COMMIT request is issued by a server to the XATMI-PM when all local resources (and all subordinate XATMI-PMs) are in the Ready state and the server is available to start the second phase of commitment.

An XATMI-COMMIT indication received by the client or the server from the XATMI-PM indicates that the second phase of commitment has been started and that the client (or the server) must commit any local resources.

After receiving this event, the client or the server does not receive a XATMI-ROLLBACK indication. The client or the server are required to issue an XATMI-DONE request to confirm that all local resources have been placed into the final (Committed) state.

There is no mapping of the XATMI-COMMIT service to the XATMI interface.

12.4.11 XATMI-DONE request

Parameters

Parameter Name	req
Heuristic-Report	U

Heuristic-Report

This parameter is supplied by the server to indicate that a heuristic decision has been made and is to be reported to the client. This parameter can have one of the following values:

- HEURISTIC-MIX: the data handled by the server are in a state that is inconsistent with the outcome of the transaction.
- HEURISTIC-HAZARD: a communication failure has occurred that may prevent the reporting of data inconsistency.

Usage

The XATMI-DONE request is a *global* transaction service that is issued by a client or a server to indicate that all local resources associated with the transaction have been placed in either their initial or final state (rolled-back or committed respectively).

This request may be issued in:

- response to an XATMI-COMMIT indication to indicate the end of the local commitment procedure
- response to an XATMI-ROLLBACK indication to indicate the end of the local rollback procedure
- response to an XATMI-CANCEL indication.

There is no mapping of the XATMI-COMMIT service to the XATMI interface.

12.4.12 XATMI-COMPLETE indication

Parameters

None.

Usage

An XATMI-COMPLETE indication is a *global* transaction service that is received by a client or a server from their XATMI-PM to indicate the completion of the transaction commitment or rollback process and therefore it signals the end of the transaction.

12.4.13 XATMI-ROLLBACK request and indication

Parameters

None.

Usage

An XATMI-ROLLBACK request is a *global* transaction service that is issued by the client to the XATMI-PM to start the rollback of the transaction. This request is mapped from *tx_rollback*( ) (see Section 3.7.1 on page 19 and the **TX** (Transaction Demarcation) specification for the rules governing the use of this function).

An XATMI-ROLLBACK indication is received from the XATMI-PM to indicate that a transaction is to be rolled back.

The client or the server is required to issue an XATMI-DONE request to confirm that all local resources have been placed into the final (Rolled-back) state.

12.4.14 XATMI-HEURISTIC indication

Parameters

Parameter Name	ind
Diagnostic	M

Diagnostic

This parameter is supplied by the XATMI-PM to indicate that a heuristic decision has been made. This parameter can have one one of the following values:

- HEURISTIC-MIX: the data handled by the XATMI-AE user is in a state that is inconsistent with the outcome of the transaction.
- HEURISTIC-HAZARD: a communication failure has occurred that may prevent the reporting of data inconsistency.

Usage

An XATMI-HEURISTIC indication is issued by the XATMI-PM to the client to indicate that a heuristic decision has been made on the transaction.

The client may map this indication to a return code in *tx_commit*( ) or *tx_rollback*( ).

An XATMI-HEURISTIC indication relates to a particular branch on the transaction tree and, usually, the XATMI-PM or the XATMI provider should record this information.

12.5 Sequencing Rules and State Table

The XATMI-ASE Service State Table is shown in Section 12.5.5 on page 143. It applies to all XATMI-SUIs.

12.5.1 State Table Conventions

The XATMI-ASE Service State Table describes the allowed sequence of service events between an XATMI-SUI and an XATMI-PM.

In the state table, each column (except **Service** and **Preconditions**) represents a state; each row represents an XATMI-ASE service primitive; and each cell represents a state transition.

The state table specifies predicates or preconditions that must be satisfied in order for individual XATMI-ASE service primitives to be valid in a given state. These predicates are based on the values of variables (see Section 12.5.3). If a variable is listed in the state table prefixed by ¬ (logical NOT), then the value of that variable must be False for the predicated transition to occur.

The state table also specifies actions to be performed. These actions involve setting variables to the specified value (see Section 12.5.4 on page 143).

12.5.2 States

The valid states are as follows:

S_0 Idle (**IDLE**): No pending events. If the the XATMI-SUI is within a transaction, then transaction completion events may be pending.

S_1 Reply expected (**RPLY**): A reply is expected from the application service.

S_2 Has Control (**CTRL**): The XATMI-SUI has control of the conversation with a conversational service.

S_3 Has No Control (¬**CTRL**): The XATMI-SUI does not have control of the conversation with a conversational service.

S_4 Start Commit Phase 1 (**SPC**): The Precommit phase has been started.

S_5 End Commit Phase 1 (**EPC**): The Precommit phase has ended.

S_6 Start Commit Phase 2 (**SC**): The Commit phase has been started.

S_7 End Commit Phase 2 (**EC**): Commitment has successfully completed.

S_8 Rollback in Progress (**RBP**): The transaction is being rolled back.

S_9 Rollback Completed (**ERB**): The transaction has been successfully rolled back.

12.5.3 Variables

The following variables are used for preconditions and state transitions:

Clnt: When this variable is **True**, the XATMI-SUI is a client or a superior node in the transaction tree (or both).

Svr: When this variable is **True**, the XATMI-SUI is a server or a subordinate node in the transaction tree (or both).

Reply: When this variable is **True**, the XATMI-SUI should expect a reply from the application service.

Conv: When this variable is **True**, the XATMI-SUI has a connection with a conversational service.

Ctrl: When this variable is **True**, the XATMI-SUI has control of the conversation with a conversational service.

Tran: When this variable is **True**, the XATMI-SUI should be participating in a global transaction.

12.5.4 Actions

[A1] Set *Conv* to **True**.

[A2] If Grant-Control is **True**, set *Ctrl* to **False**.

[A3] If Grant-Control is **True**, set *Ctrl* to **True**.

12.5.5 State Table

The XATMI-ASE Service State Table is as follows:

Service	**Preconditions**	**IDLE** S_0	**RPLY** S_1	**CTRL** S_2	**¬CTRL** S_3	**SPC** S_4	**EPC** S_5	**SC** S_6	**EC** S_7	**RBP** S_8	**ERB** S_9
XATMI-CALL req	*Clnt, Reply, ¬Tran*	S_1									
	Clnt, Reply, Tran	S_1									
	Clnt, ¬Reply, ¬Tran	S_0									
XATMI-CALL ind	*Svr, Reply*	S_1									
	Svr, ¬Reply, ¬Tran	S_0									
XATMI-REPLY req	*Svr, Reply*		S_0								
	Svr, Conv			S_0	S_0						
XATMI-REPLY ind	*Clnt, Reply*		S_0								
	Clnt, Conv			S_0	S_0						
XATMI-FAILURE req	*Svr, Reply, ¬Tran*		S_0								
	Svr, Conv, ¬Tran			S_0	S_0						
	Svr, Reply, Tran		S_8								
	Svr, Conv, Tran			S_8	S_8						
XATMI-FAILURE ind	*Clnt, Reply, ¬Tran*		S_0								
	Clnt, Conv, ¬Tran			S_0	S_0						
	Clnt, Reply, Tran		S_8								
	Clnt, Conv, Tran			S_8	S_8						
XATMI-CANCEL req	*Clnt, Reply, ¬Tran*		S_0								
XATMI-CANCEL ind	*Svr, Reply, ¬Tran*		S_0								
XATMI-CONNECT req	*Clnt, Ctrl*	[A1] S_2									
	Clnt, ¬Ctrl	[A1] S_3									

Service	**Preconditions**	**IDLE** S_0	**RPLY** S_1	**CTRL** S_2	**¬CTRL** S_3	**SPC** S_4	**EPC** S_5	**SC** S_6	**EC** S_7	**RBP** S_8	**ERB** S_9
XATMI-CONNECT ind	*Svr, Ctrl*	[A1] S_2									
	Svr, ¬Ctrl	[A1] S_3									
XATMI-DISCON req	*Clnt, Conv, ¬Tran*			S_0	S_0						
	Clnt, Conv, Tran			S_8	S_8						
XATMI-DISCON ind	*Svr, Conv, ¬Tran*			S_0	S_0						
	Svr, Conv, Tran			S_8	S_8						
XATMI-DATA req	*Conv, Ctrl*			[A2] S_3							
	Conv, Ctrl			S_2							
XATMI-DATA ind	*Conv, ¬Ctrl*				[A3] S_2						
	Conv, ¬Ctrl				S_3						
XATMI-PREPARE req	*Clnt, Tran*	S_4									
XATMI-PREPARE ind	*Svr, Tran*	S_4									
XATMI-READY ind	*Clnt, Tran*					S_5					
XATMI-COMMIT req	*Svr, Tran*					S_5					
	Clnt, Tran						S_5				
XATMI-COMMIT ind	*Tran*						S_6				
XATMI-DONE req	*Tran*							S_7		S_9	
XATMI-COMPLETE ind	*Tran*								S_0		S_0
XATMI-ROLLBACK req	*Clnt, Tran*	S_8	S_8	S_8	S_8	S_8	S_8				
	Svr, Tran					S_8					
XATMI-ROLLBACK ind	*Clnt, Tran*	S_8	S_8				S_8				
	Svr, Tran	S_8	S_8	S_8	S_8	S_8	S_8				
XATMI-HEURISTIC ind	*Clnt, Tran*							S_6	S_7	S_8	

Table 12-4 XATMI-ASE Service State Table

Note: Transaction services apply to all XATMI-SUIs within a transaction. Also, notice that the root of the transaction tree (from the OSI TP point of view) is an XATMI-SUI that behaves as a client.

Chapter 13

XATMI-ASE Protocol Specification

This chapter outlines the relationship between XATMI services and other ASEs, and summarises the relevant mappings. It also provides mapping details for each XATMI-ASE, from the XATMI interface and to the OSI TP services.

13.1 Relationship with Other ASEs

The XATMI services provided by the XATMI Protocol Machine (XATMI-PM) are mapped onto services provided by the OSI TPPM (see the OSI TP Protocol standard). OSI TP concatenation rules apply and are assumed to be enforced by the TPPM.

The following conventions are used to describe the corresponding service mappings:

- An XATMI-PM may take a *client role* or a *server role*. The XATMI-PM role corresponds to the XATMI-SUIs described in Chapter 12.
- The abstract service TP-DATA is used to represent the mapping of XATMI APDUs to OSI TP according to OSI TP concatenation rules.
- The abstract service TP-PREPARE-ALL request is used as defined in the **XAP-TP** specification. The TP-PREPARE-ALL request service performs the first phase of commitment for a transaction (that is, this service represents the combination of the TP-PREPARE requests issued on every dialogue associated with the transaction).
- The abstract service TP-READY-ALL indication is used as defined in the **XAP-TP** specification. The TP-READY-ALL indication informs the completion of the first phase of commitment (that is, this service represents the combination of all TP-READY indications received from the subordinate dialogues and the TPPM changing the transaction state to **Ready**).
- The abstract service TP-COMMIT-ALL request is not the service TP-COMMIT request defined in the OSI TP Protocol standard, but it is an instruction to the OSI TP MACF to start the second phase of commitment. The combination of the TP-PREPARE-ALL request and TP-COMMIT-ALL request used throughout this document, is the equivalent of the OSI TP abstract service TP-COMMIT as defined in the OSI TP Service standard. The abstract service TP-COMMIT-ALL request is used as defined by the TP-COMMIT request in the referenced **XAP-TP** specification. The term TP-COMMIT-ALL is used in place of the term TP-COMMIT to avoid confusion for readers familiar with the OSI TP standards.
- The service TP-DEFERRED-END-DIALOGUE request is used when a dialogue has been started within a transaction to limit the lifetime of the dialogue to the duration of the transaction. For clarity, the mapping to this service is shown after a TP-BEGIN-DIALOGUE request but this service could also be issued from other mappings, for example from an XATMI-PREPARE request. The main requirement for an XATMI-PM taking a server role is that a TP-DEFERRED-END-DIALOGUE indication must have been received before a TP-PREPARE indication.
- Some of the mappings (for example MAP 10) assume that the TPPM provider offers a service to end a concatenation and to force the delivery of the corresponding APDUs to the remote TPSUI.

13.2 Client Role Mappings

The following table summarises the client role mappings.

Each mapping is identified with a number provided in the **Map No.** column. These numbers are used in later sections that describe these mappings in more detail.

Mappings enclosed in brackets ([ ]) are conditional, that is they are generated dependent on the variables used with the corresponding XATMI-ASE service.

Table 13-1 Client Role Mappings

XATMI Services (Client)	Map No.	See:	TP/Presentation Services
XATMI-CALL req	1	Section 13.6.2 on page 155.	TP-BEGIN-DIALOGUE req [TP-DEFER-END-DIALOGUE req] TP-DATA req TP-GRANT-CONTROL req
	2	Section 13.6.2 on page 155.	TP-BEGIN-DIALOGUE req TP-DATA req TP-END-DIALOGUE req
XATMI-REPLY ind	3	Section 13.15.2 on page 169.	TP-DATA ind TP-GRANT-CONTROL ind
	4	Section 13.15.2 on page 169.	TP-DATA ind TP-END-DIALOGUE ind
XATMI-FAILURE ind	5	Section 13.16.2 on page 171.	TP-U-ABORT ind
	6	Section 13.16.2 on page 171.	TP-P-ABORT ind
	7	Section 13.16.2 on page 171.	TP-BEGIN-DIALOGUE(Reject) cnf
XATMI-CANCEL req	8	Section 13.10.2 on page 161.	TP-U-ABORT req
XATMI-CONNECT req	9	Section 13.7.2 on page 158.	TP-BEGIN-DIALOGUE req [TP-DEFER-END-DIALOGUE req] TP-DATA req TP-GRANT-CONTROL req
	10	Section 13.7.2 on page 158.	TP-BEGIN-DIALOGUE req [TP-DEFER-END-DIALOGUE req] TP-DATA req TP-BEGIN-DIALOGUE cnf
XATMI-DISCON req	11	Section 13.12.2 on page 163.	TP-U-ABORT req

XATMI Services (Client)	Map No.	See:	TP/Presentation Services
XATMI-DATA req	12	Section 13.11.2 on page 162.	TP-DATA req [TP-GRANT-CONTROL req]
XATMI-DATA ind	13	Section 13.19.2 on page 177.	TP-DATA ind [TP-GRANT-CONTROL ind]
XATMI-PREPARE req	14	**Mapping to OSI TP** on page 178.	TP-PREPARE-ALL req
XATMI-READY ind	15	**Mapping from OSI TP** on page 181.	TP-READY-ALL ind
XATMI-COMMIT req	16	**Mapping to OSI TP** on page 178.	TP-COMMIT-ALL req
XATMI-COMMIT ind	17	**Mapping from OSI TP** on page 181.	TP-COMMIT ind
XATMI-DONE req	18	Section 13.20.3 on page 179.	TP-DONE req
	19	Section 13.20.3 on page 179.	TP-U-ABORT req TP-DONE req
XATMI-ROLLBACK req	20	**Mapping to OSI TP** on page 180.	TP-ROLLBACK req
XATMI-ROLLBACK ind	21	**Mapping from OSI TP** on page 181.	TP-P-ABORT ind
	22	**Mapping from OSI TP** on page 181.	TP-U-ABORT ind
XATMI-COMPLETE ind	23	**Mapping from OSI TP** on page 182.	TP-COMMIT-COMPLETE ind
	24	**Mapping from OSI TP** on page 182.	TP-ROLLBACK-COMPLETE ind
XATMI-HEURISTIC ind	25	**Mapping from OSI TP** on page 182.	TP-HEURISTIC-REPORT ind

13.3 Server Role Mappings

The following table summarises the server role mappings.

The client and server mappings are numbered consecutively to allow unique references throughout the document.

Table 13-2 Server Role Mappings

XATMI Services (Server)	Map No.	See:	TP/Presentation Services
XATMI-CALL ind	26	Section 13.13.2 on page 164.	TP-BEGIN-DIALOGUE ind [TP-DEFER-END-DIALOGUE ind] TP-DATA ind TP-GRANT-CONTROL ind
	27	Section 13.13.2 on page 164.	TP-BEGIN-DIALOGUE ind TP-DATA ind TP-END-DIALOGUE ind
	28	Section 13.13.2 on page 164.	TP-BEGIN-DIALOGUE ind TP-BEGIN-DIALOGUE rsp
XATMI-CONNECT ind	29	Section 13.14.2 on page 167.	TP-BEGIN-DIALOGUE ind [TP-DEFER-END-DIALOGUE ind] TP-DATA ind TP-GRANT-CONTROL ind
	30	Section 13.14.2 on page 167.	TP-BEGIN-DIALOGUE ind [TP-DEFER-END-DIALOGUE ind] TP-DATA ind TP-BEGIN-DIALOGUE rsp
	31	Section 13.14.2 on page 167.	TP-BEGIN-DIALOGUE ind TP-BEGIN-DIALOGUE rsp
XATMI-REPLY req	32	Section 13.8.2 on page 159.	TP-DATA req TP-GRANT-CONTROL req
	33	Section 13.8.2 on page 159.	TP-DATA req TP-END-DIALOGUE req
XATMI-FAILURE req	34	Section 13.9.2 on page 160.	TP-U-ABORT req
XATMI-CANCEL ind	35	Section 13.17.2 on page 175.	TP-U-ABORT ind
	36	Section 13.17.2 on page 175.	TP-P-ABORT ind

XATMI Services (Server)	Map No.	See:	TP/Presentation Services
XATMI-DISCON ind	37	Section 13.18.2 on page 176.	TP-U-ABORT ind
	38	Section 13.18.2 on page 176.	TP-P-ABORT ind
XATMI-DATA ind	39	Section 13.19.2 on page 177.	TP-DATA ind [TP-GRANT-CONTROL ind]
XATMI-DATA req	40	Section 13.11.2 on page 162.	TP-DATA req [TP-GRANT-CONTROL req]
XATMI-PREPARE ind	41	**Mapping from OSI TP** on page 180.	TP-PREPARE-ALL ind
XATMI-COMMIT req	42	**Mapping to OSI TP** on page 178.	TP-COMMIT-ALL req
XATMI-COMMIT ind	43	**Mapping from OSI TP** on page 181.	TP-COMMIT ind
XATMI-DONE req	44	Section 13.20.3 on page 179.	TP-DONE req
	45	Section 13.20.3 on page 179.	[TP-U-ABORT req] TP-DONE req
XATMI-ROLLBACK req	46	**Mapping to OSI TP** on page 180.	TP-U-ABORT req
XATMI-ROLLBACK ind	47	**Mapping from OSI TP** on page 181.	TP-ROLLBACK ind
	48	**Mapping from OSI TP** on page 181.	TP-P-ABORT ind
XATMI-COMPLETE ind	49	**Mapping from OSI TP** on page 182.	TP-COMMIT-COMPLETE ind
	50	**Mapping from OSI TP** on page 182.	TP-ROLLBACK-COMPLETE ind

13.4 OSI TP Services Used by the XATMI-ASE

The following table presents the OSI TP services used by the XATMI-ASE:

Services	req	ind	rsp	cnf
TP-BEGIN-DIALOGUE	•	•	•	•
TP-END-DIALOGUE	•	•	×	×
TP-U-ABORT	•	•		
TP-P-ABORT		•		
TP-U-ERROR	×	×		
TP-GRANT-CONTROL	•	•		
TP-REQUEST-CONTROL	×	×		
TP-HANDSHAKE	×	×	×	×
TP-HANDSHAKE-AND-GRANT-CONTROL	×	×	×	×
TP-BEGIN-TRANSACTION	×	×		
TP-DEFERRED-END-DIALOGUE	•	•		
TP-DEFERRED-GRANT-CONTROL	×	×		
TP-PREPARE	$\underline{\bullet}$	•		
TP-READY		$\underline{\bullet}$		
TP-COMMIT	$\underline{\bullet}$	•		
TP-DONE	•			
TP-COMMIT-COMPLETE		•		
TP-ROLLBACK	•	•		
TP-ROLLBACK-COMPLETE		•		
TP-HEURISTIC-REPORT		•		

Table 13-3 OSI TP Services Used by the XATMI-ASE

Key:

• Used directly by the XATMI-ASE.

$\underline{\bullet}$ Used indirectly by the XATMI-ASE. These services are generated by the implementation of the abstract service TP-PREPARE-ALL request, TP-READY-ALL indication, and TP-COMMIT-ALL request.

× Not used by the XATMI-ASE.

13.5 Summary of Mappings between OSI TP and XATMI-ASE

OSI TP Service/Service Abstraction	req	ind
TP-BEGIN-DIALOGUE	XATMI-CALL req XATMI-CONNECT req	XATMI-CALL ind XATMI-CONNECT ind
TP-END-DIALOGUE	XATMI-CALL req XATMI-REPLY req	XATMI-CALL ind XATMI-REPLY ind
TP-U-ABORT	XATMI-FAILURE req XATMI-CANCEL req XATMI-DISCON req XATMI-ROLLBACK req XATMI-DONE req	XATMI-FAILURE ind XATMI-CANCEL ind XATMI-DISCON ind XATMI-ROLLBACK ind
TP-P-ABORT		XATMI-FAILURE ind XATMI-DISCON ind XATMI-CANCEL ind XATMI-ROLLBACK ind
TP-GRANT-CONTROL	XATMI-CALL req XATMI-CONNECT req XATMI-DATA req	XATMI-CALL ind XATMI-CONNECT ind XATMI-DATA ind
TP-DEFERRED-END-DIALOGUE	XATMI-CALL req XATMI-CONNECT req	XATMI-CALL ind XATMI-CONNECT ind
TP-PREPARE-ALL	XATMI-PREPARE req	
TP-PREPARE		XATMI-PREPARE ind
TP-READY-ALL		XATMI-READY ind
TP-COMMIT-ALL	XATMI-COMMIT req	
TP-COMMIT		XATMI-COMMIT ind
TP-DONE	XATMI-DONE req	
TP-COMMIT-COMPLETE		XATMI-COMPLETE ind
TP-ROLLBACK	XATMI-ROLLBACK req	XATMI-ROLLBACK ind
TP-ROLLBACK-COMPLETE		XATMI-COMPLETE ind
TP-HEURISTIC-REPORT		XATMI-HEURISTIC ind
TP-DATA	XATMI-CALL req XATMI-REPLY req XATMI-DATA req XATMI-CONNECT req	XATMI-CALL ind XATMI-REPLY ind XATMI-DATA ind XATMI-CONNECT ind

OSI TP Service/Service abstraction	rsp	cnf
TP-BEGIN-DIALOGUE	XATMI-CALL ind* XATMI-CONNECT ind*	XATMI-FAILURE ind XATMI-CONNECT req*
TP-END-DIALOGUE	Not used	Not used

Table 13-4 Mappings Between OSI TP and XATMI-ASE

* the implementation of the XATMI-CALL and XATMI-CONNECT indications may reject a service request with a TP-BEGIN-DIALOGUE response (see Section 13.13.2 on page 164). In this case the corresponding indication is not issued to the server. An XATMI-CONNECT request may fail if a TP-BEGIN-DIALOGUE(Reject) confirmation is received (see Section 13.7.2 on page 158).

The following sections define the mapping of the XATMI-ASE:

- from the XATMI interface
- to the OSI TP services.

13.6 XATMI-CALL request

13.6.1 Mapping from tpacall()/tpcall()

An XATMI-CALL request is mapped from *tpacall*( ) or *tpcall*( ). The following tables summarise the corresponding parameter mappings:

tpacall( )		**XATMI-CALL req**	**Notes**
svc		Service-Name	The XATMI-PM uses this name to retrieve the parameters for TP-BEGIN-DIALOGUE req from the local configuration information, and to set the value of the **service** field of the XATMI-CALL-RI APDU (see Chapter 14).
data, len		User-Data	The XATMI-PM uses the abstract syntax defined in Chapter 14 to encode the typed buffer and to set the value of the **data** field of the XATMI-CALL-RI APDU.
flags	TPNOREPLY	No-Reply-Option = True	The XATMI-PM uses this value to determine the sequencing of the TP services generated by this service, and to select the ATP-11 profile.
	TPNOTRAN	Begin-Transaction = False	The XATMI-PM uses this value to determine if the TP dialogue must be included within the current global transaction.
	TPNOBLOCK	No direct mapping	Local to each implementation.
	TPNOTIME	No direct mapping	Local to each implementation.
	TPSIGRSTRT	No direct mapping	Local to each implementation.

tpcall( )		**XATMI-CALL req**	**Notes**
svc		Service-Name	The XATMI-PM uses this name to retrieve the parameters for TP-BEGIN-DIALOGUE from the local configuration information, and to set the value of the **service** field of the XATMI-CALL-RI APDU.
idata, ilen		User-Data	The XATMI-PM uses the abstract syntax defined in Chapter 14 to encode the typed buffer and to set the value of the **data** field of the XATMI-CALL-RI APDU.
odata, olen			Mapped from XATMI-REPLY indication (Section 13.15 on page 169). or XATMI-FAILURE indication (Section 13.16 on page 170).
		No-Reply-Option = False	*tpcall*( ) always expects a reply. This reply comes with an XATMI-REPLY indication or an XATMI-FAILURE indication.
flags	TPNOTRAN	Begin-Transaction = False	The XATMI-PM uses this value to determine if the TP dialogue must be included within the current global transaction, and to select the TP functional units.
	TPNOCHANGE	No direct mapping	Local to each implementation.
	TPNOBLOCK	No direct mapping	Local to each implementation.
	TPNOTIME	No direct mapping	Local to each implementation.
	TPSIGRSTRT	No direct mapping	Local to each implementation.

13.6.2 Mapping to OSI TP

The XATMI-CALL request parameters are used by the XATMI-PM to extract the necessary information from the local configuration information for mapping to the TP-BEGIN-DIALOGUE request, and to construct an XATMI-CALL-RI APDU.

The following table summarises this mapping:

TP-BEGIN-DIALOGUE	req	XATMI-CALL req
Initiating-AP-Title	M	From local configuration information
Initiating-API Identifier	O	From local configuration information
Initiating-AE-Qualifier	M	From local configuration information
Initiating-AEI-Identifier	O	From local configuration information
Initiating-TPSU-title	U	From local configuration information
Recipient-AP-Title	M	From local configuration information
Recipient-AE-Qualifier	C	From local configuration information and is mandatory in a provider-supported transaction
Recipient-API-Identifier	U	From local configuration information
Recipient-AEI-Identifier	U	From local configuration information
Recipient-TPSU-Title	U	From local configuration information (or value of Service-Name)
Quality-of-Service	U	From local configuration information
Application-Context-Name	M	Set to XATMI Application Context identifier (see Chapter 11)
Begin-Transaction ={	C	From Begin-Transaction if ATP21; otherwise, not present
True		True
False}		False}
Confirmation = {	M	Always set to Negative
Always		
Negative}		
Functional Units = {	M	From Begin-Transaction and the local configuration information
Dialogue		Always used (ATP11, ATP21, ATP31)
Polarized Control		Always used (ATP11, ATP21, ATP31)
Shared Control		Not used
Commit		Used if Begin-Transaction = True (ATP21 or ATP31)
Unchained Transactions		Used if Begin-Transaction = True and ATP21
Chained Transactions		Used if Begin-Transaction = True and ATP31
Handshake}		Not Used
User-Data	U	Not used

There are two basic mappings of the XATMI-CALL request onto OSI TP services:

MAP 1: This mapping is generated when the No-Reply-Option parameter is set to **False**:

1. A TP-BEGIN-DIALOGUE request is issued as specified above, followed by a TP-DEFERRED-END-DIALOGUE request if the Begin-Transaction parameter is set to **True**. Note that an implementation may delay sending the TP-DEFERRED-END-DIALOGUE request until just before an XATMI-PREPARE request is issued.
2. The Service-Name parameter is encoded into the **service** field of the XATMI-CALL-RI APDU as specified in Chapter 14. If the User-Data parameter is specified, the corresponding typed buffer is encoded into the **data** field of the same APDU. The APDU is mapped to the abstract service TP-DATA request.
3. A TP-GRANT-CONTROL request is then issued.

MAP 2: This mapping is generated when the No-Reply-Option parameter is set to **True**. In this case the Begin-Transaction parameter must be set to **False**:

1. A TP-BEGIN-DIALOGUE request is issued as specified above.
2. The Service-Name parameter is encoded into the **service** field of the XATMI-CALL-RI APDU as specified in Chapter 14. If the User-Data parameter is specified, the corresponding typed buffer is encoded into the **data** field of the same APDU. The APDU is mapped to the abstract service TP-DATA request.
3. A TP-END-DIALOGUE request is then issued.

13.7 XATMI-CONNECT request

13.7.1 Mapping from tpconnect( )

An XATMI-CONNECT request is mapped from *tpconnect*( ). The following table summarises the corresponding parameter mapping:

tpconnect( )		**XATMI-CONNECT req**	**Notes**
svc		Service-Name	The XATMI-PM uses this name to retrieve the parameters for TP-BEGIN-DIALOGUE from the local configuration information and to set the value of the **service** field of the XATMI-CONNECT-RI APDU.
data, len		User-Data	The XATMI-PM uses the abstract syntax defined in Chapter 14 to encode the typed buffer and to set the value of the **data** field of the XATMI-CONNECT-RI APDU.
flags	TPNOTRAN	Begin-Transaction = False	The XATMI-PM uses this value to determine if the TP dialogue must be included within the current global transaction, and to select the TP functional units.
	TPSENDONLY	Grant-Control = False	The XATMI-PM uses this value to generate the TP protocol necessary for the confirmation of the dialogue
	TPRECVONLY	Grant-Control = True	The XATMI-PM uses this value to grant control of the dialogue to remote service.
	TPNOBLOCK	No direct mapping	Local to each implementation.
	TPNOTIME	No direct mapping	Local to each implementation.
	TPSIGRSTRT	No direct mapping	Local to each implementation.

13.7.2 Mapping to OSI TP

The XATMI-CONNECT request parameters are used by the XATMI-PM to extract the necessary information from the local configuration information for the mapping to the TP-BEGIN-DIALOGUE request, and to construct an XATMI-CONNECT-RI APDU.

The table defined in Section 13.6.2 on page 155 for the mapping of an XATMI-CALL request to a TP-BEGIN-DIALOGUE request also applies to the XATMI-CONNECT request mapping. The only difference is that the Confirmation parameter of the TP-BEGIN-DIALOGUE request may in some cases be set to **Always**.

The following mappings to OSI TP are defined:

MAP 9: This mapping is used when the Grant-Control parameter is set to **True**.

1. A TP-BEGIN-DIALOGUE request is issued as specified above (see MAP 1:1, Section 13.6.2 on page 155), followed by a TP-DEFERRED-END-DIALOGUE request if the Begin-Transaction parameter is set to **True**. Note that an implementation may delay sending the TP-DEFERRED-END-DIALOGUE request until just before an XATMI-PREPARE request is issued.
2. The Service-Name parameter is encoded into the **service** field of the XATMI-CONNECT-RI APDU as specified in Chapter 14. If the User-Data parameter has been specified, then the corresponding typed buffer is encoded into the **data** field of the same APDU. Then the APDU is mapped to a TP-DATA request.
3. A TP-GRANT-CONTROL request is then issued.

MAP 10: This mapping is used when the Grant-Control parameter is set to **False**.

1. A TP-BEGIN-DIALOGUE request is issued as specified above (see MAP 1:1 Section 13.6.2 on page 155), but with the Confirmation parameter set to **Always**, followed by a TP-DEFERRED-END-DIALOGUE request if the Begin-Transaction parameter is set to **True**. Note that an implementation may delay sending the TP-DEFERRED-END-DIALOGUE request until just before an XATMI-PREPARE request is issued.
2. The Service-Name parameter is encoded into the **service** field of the XATMI-CONNECT-RI APDU as specified in Chapter 14. If the User-Data parameter has been specified, the corresponding typed buffer is encoded into the **data** field of the same APDU. Then the APDU is mapped to a TP-DATA request and the XATMI-PM issues the TPPM **flush** instruction to ensure that the buffered APDUs are sent to their remote destinations.
3. The XATMI-CONNECT request implementation should wait for a TP-BEGIN-DIALOGUE confirm indicating the success of the conversation establishment with the application service. If a TP-BEGIN-DIALOGUE confirmation with Result set to Rejected is received, the XATMI-CONNECT request fails. It also fails if a TP-P-ABORT indication is received.

13.8 XATMI-REPLY request

13.8.1 Mapping from tpreturn( )

An XATMI-REPLY request is mapped from *tpreturn*( ) with the *rval* parameter set to TPSUCCESS. The following table defines the parameter mapping:

tpreturn( )	**XATMI-REPLY req**	**Notes**
rval = TPSUCCESS		The XATMI-PM uses this value to map the application service reply to this service.
rcode	User-Code	The XATMI-PM includes this value in the XATMI-REPLY-RI APDU that is generated according to the rules defined in Chapter 14.
data, len	User-Data	The XATMI-PM uses the abstract syntax defined in Chapter 14 to encode the typed buffer and to set the value of the **data** field of the XATMI-REPLY-RI APDU.
flags		*tpreturn*( ) has no flags defined.

13.8.2 Mapping to OSI TP

An XATMI-REPLY request is mapped as follows:

MAP 32: This mapping is used when a service request is invoked within a global transaction.

1. An XATMI-REPLY-RI APDU as defined in Chapter 14, is generated by the XATMI-PM. This APDU, which contains the User-Code and the User-Data parameters, is mapped to a TP-DATA request.
2. A TP-GRANT-CONTROL request is then issued.

MAP 33: This mapping is used when a service request is not invoked within a global transaction.

1. An XATMI-REPLY-RI APDU (as defined in Chapter 14) is generated by the XATMI-PM. This APDU, which contains the User-Code and the User-Data parameters, is mapped to a TP-DATA request.
2. A TP-END-DIALOGUE request is then issued.

13.9 XATMI-FAILURE request

13.9.1 Mapping from tpreturn( )

An XATMI-FAILURE request is mapped from *tpreturn*( ) with the *rval* parameter set to TPFAIL. The following table defines the parameter mapping:

tpreturn( )	**XATMI-FAILURE req**	**Notes**
rval = TPFAIL	Diagnostic = Application-Service-Failure	The XATMI Provider uses this *rval* value to map the service reply to this XATMI-ASE service. The XATMI-PM uses the Diagnostic value to set the value of the **diagnostic** field of the XATMI-FAILURE-RI APDU according to the rules defined in Chapter 14. Note that Diagnostic may be changed to **Recipient-XATMI-SU-Failure** if the XATMI Provider finds an error during the processing of *tpreturn*( ).
rcode	User-Code	The XATMI-PM uses this value to set the **user-code** field of the XATMI-FAILURE-RI APDU according to the rules defined in Chapter 14.
data, len	User-Data	The XATMI-PM uses the abstract syntax defined in Chapter 14 to encode the typed buffer and to set the value of the **data** field of the XATMI-FAILURE-RI APDU. This APDU is sent with a TP-U-ABORT request as defined below.
flags		*tpreturn*( ) has no flags defined.

13.9.2 Mapping to OSI TP

An XATMI-FAILURE request is mapped as follows:

MAP 34: This mapping is generated from *tpreturn*( ). The XATMI-PM generates an XATMI-FAILURE-RI APDU and maps it to the User-Data parameter of the TP-U-ABORT request. The XATMI-PM sets the **diagnostic**, **user-code** and **data** fields of the XATMI-FAILURE-RI APDU as defined in Section 13.9.1.

Note: If the XATMI-FAILURE request initiates rollback of the current transaction, any TP-P-ABORT indication or TP-U-ABORT indication primitives received by the XATMI-PM between the XATMI-FAILURE request and the subsequent XATMI-DONE request are silently discarded by the XATMI-PM. TP-U-ABORT requests for any remaining dialogues take place when the XATMI-DONE request is issued.

13.10 XATMI-CANCEL request

13.10.1 Mapping from tpcancel( )

An XATMI-CANCEL request is mapped from *tpcancel*( ). There are no parameters to be mapped.

13.10.2 Mapping to OSI TP

An XATMI-CANCEL request is mapped as follows:

MAP 8: The client XATMI-PM maps the XATMI-CANCEL request to a TP-U-ABORT request. The User-Data parameter of the TP-U-ABORT service is not used.

13.11 XATMI-DATA request

13.11.1 Mapping from tpsend( )

An XATMI-DATA request is mapped from *tpsend*( ). The following table summarises the corresponding parameter mapping:

tpsend( )		**XATMI-DATA req**	**Notes**
cd			Local processing.
data, len		User-Data	The XATMI-PM uses the abstract syntax defined in Chapter 14 to encode the typed buffer and to set the value of the **data** field of either an XATMI-DATA-RI or XATMI-DATA-GRANT-CONTROL-RI APDU. XATMI-DATA-GRANT-CONTROL-RI is used if the TPRECVONLY flag is set, and XATMI-DATA-RI otherwise.
flags	TPRECVONLY	Grant-Control = True	The XATMI-PM uses this value to grant control of the dialogue to the remote service.
	TPNOBLOCK	No direct mapping	Local to each implementation.
	TPNOTIME	No direct mapping	Local to each implementation.
	TPSIGRSTRT	No direct mapping	Local to each implementation.
rvent	TPEV_DISCONIMM		Mapped from an XATMI-DISCON indication or an XATMI-FAILURE indication.
	TPEV_SVCFAIL		Mapped from an XATMI-FAILURE indication.
	TPEV_SCVERR		Mapped from an XATMI-FAILURE indication.

13.11.2 Mapping to OSI TP

An XATMI-DATA request is mapped as follows:

MAP 12: The client XATMI-PM maps an XATMI-DATA request as follows:

1. If the User-Data parameter contains a typed buffer, it is encoded as described in Chapter 14 and mapped to the **data** field of an XATMI-DATA-GRANT-CONTROL-RI APDU if the TPRECVONLY flag is set, and to XATMI-DATA-RI APDU otherwise. The APDU is then mapped to a TP-DATA request.

2. A TP-GRANT-CONTROL request is issued if the Grant-Control parameter is set to **True**.

MAP 40: This mapping is the same as Map 12. However, it is generated from the server XATMI-PM.

13.12 XATMI-DISCON request

13.12.1 Mapping from tpdiscon()

An XATMI-DISCON request is mapped from *tpdiscon*( ). There are no parameters to be mapped.

13.12.2 Mapping to OSI TP

An XATMI-DISCON request is mapped as follows:

MAP 11: The XATMI-PM maps the XATMI-DISCON request to a TP-U-ABORT request. The User-Data parameter of the TP-U-ABORT service is not used.

Note: If the XATMI-DISCON request initiates rollback of the current transaction, any TP-P-ABORT indication or TP-U-ABORT indication primitives received by the XATMI-PM between the XATMI-DISCON request and the subsequent XATMI-DONE request are silently discarded by the XATMI-PM. TP-U-ABORT requests for any remaining dialogues take place when the XATMI-DONE request is issued.

13.13 XATMI-CALL indication

13.13.1 Mapping to tpservice( )

An XATMI-CALL indication is issued to the server by the XATMI-PM on receipt of a request to a request/response service. The server uses the *tpservice*( ) template to map the parameters and then dispatches the corresponding application routine.

The *tpservice*( ) template is mapped from the XATMI-CALL parameters as follows:

tpservice( )		**XATMI-CALL ind**	**Notes**
cd			Local processing — not used in request/response services.
name		Service-Name	The Service-Name is mapped from the **service** field of the XATMI-CALL-RI APDU.
data, len		User-Data	The XATMI-PM uses the abstract syntax defined in Chapter 14 to decode the typed buffer from the **data** field of the XATMI-CALL-RI APDU.
flags	TPTRAN	Begin-Transaction = True	The service must be invoked as part of the client's transaction.
	TPNOREPLY	No-Reply-Option = True	The caller is not expecting a reply from the application service.
	TPCONV		Not permitted for request/response services.
	TPSENDONLY		Not permitted for request/response services.
	TPRECVONLY		Not permitted for request/response services.

13.13.2 Mapping from OSI TP

The XATMI-CALL indication parameters are generated according to the particular TP-ASE mapping used by the client XATMI-PM.

The XATMI-CALL indication parameters are generated as follows:

- The Service-Name parameter is mapped from the **service** field of the XATMI-CALL-RI APDU received with the TP-DATA indication (see below).
- The No-Reply-Option parameter is set to **True** if MAP 2 is used by the client XATMI-PM (that is, a TP-END-DIALOGUE indication is received).
- The Begin-Transaction parameter is set to **True** if the Chained Transaction functional unit is selected or if the Unchained Transaction functional unit is selected and the Begin-Transaction parameter in the TP-BEGIN-DIALOGUE indication is set to **True**.
- The User-Data parameter is set if the **data** field of the XATMI-CALL-RI APDU is present. This field is decoded into a local typed buffer structure following the rules described in Chapter 14.

The following is a description of the different mappings:

MAP 26: This mapping is normally used when a client XATMI-PM requires a reply from the application service.

1. When a TP-BEGIN-DIALOGUE indication is received by the server XATMI-PM, an XATMI-CALL-invocation procedure is started. This procedure performs validations against the local configuration information, and if any validation fails MAP 28 is then performed. The parameters of the TP-BEGIN-DIALOGUE indication are mapped as described above. If the Begin-Transaction parameter in the XATMI-CALL indication is set to **True**, the XATMI-PM includes this request within the context of the client's global transaction. The parameters from the TP-BEGIN-DIALOGUE indication are mapped as follows:

TP-BEGIN-DIALOGUE	ind	XATMI-CALL ind
Initiating-AP-Title	O	May be used for validation purposes.
Initiating-API-Identifier	O	Not used.
Initiating-AE-Qualifier	O	May be used for validation purposes.
Initiating-AEI-Identifier	O	Not Used.
Initiating-TPSU-Title	O	May be used for validation purposes.
Functional Units	M	Contains settings from the requester. Begin-Transaction is set to **True** if the Chained Transaction functional unit is selected.
Begin-Transaction	C	Begin-Transaction is set to this value if the Unchained Transaction functional unit is selected.
Confirmation	M	Always set to Negative.
User-Data	U	Not used.

2. A TP-DEFERRED-END-DIALOGUE indication is received only when the dialogue is within the context of a global transaction. No action is taken by the server XATMI-PM. Note that this indication may be received at any time until an XATMI-PREPARE-RI APDU is received.

3. An XATMI-CALL-RI APDU is then received. This APDU is decoded following the rules specified in Chapter 14. The Service-Name and User-Data parameters of the XATMI-CALL indication are set from the values of the **service** and **data** fields of the XATMI-CALL-RI APDU.

4. A TP-GRANT-CONTROL indication marks the completion of the XATMI-CALL invocation procedure. The XATMI-PM issues an XATMI-CALL indication to the server. The server translates this indication into an invocation to the corresponding application service.

MAP 27: This mapping is used when the client issues a service request that requires no reply.

1. Upon receiving the TP-BEGIN-DIALOGUE indication, the XATMI-PM starts an XATMI-CALL indication procedure. This procedure maps the TP-BEGIN-DIALOGUE indication parameters as specified above.

2. An XATMI-CALL-RI APDU is then received. This APDU is decoded following the rules specified in Chapter 14. The Service-Name and User-Data parameters of the XATMI-CALL indication are set from the values of the **service** and **data** fields of the XATMI-CALL-RI APDU.

3. When a TP-END-DIALOGUE indication is received, the XATMI-PM sets the No-Reply-Option to **True**, and completes the XATMI-CALL-invocation procedure. The XATMI-PM issues an XATMI-CALL indication to the server, and the server invokes the identified application routine.

MAP 28: This mapping is used to reject a service request.

1. When a TP-BEGIN-DIALOGUE indication is received, the XATMI-PM starts the XATMI-CALL invocation procedure that validates the corresponding parameters. If any validation fails, the procedure rejects the dialogue with a TP-BEGIN-DIALOGUE response.

2. A TP-BEGIN-DIALOGUE response is issued with the parameter mapping specified in the table below. The XATMI-CALL invocation procedure ends at this point. Notice that a TP-BEGIN-DIALOGUE may also be rejected by the TP-ASE provider (see MAP 7, Section 13.16.2 on page 171).

   The mapping of the TP-BEGIN-DIALOGUE response parameters is as follows:

TP-BEGIN-DIALOGUE	**rsp**	**XATMI-CALL ind Procedure**
Result = {	M	
Accepted		Not used — Dialogues are not confirmed.
Rejected(user)}		Maps 28, 31 — Service request rejected.
User Data	U	Not used.

13.14 XATMI-CONNECT indication

13.14.1 Mapping to tpservice( )

An XATMI-CONNECT indication is issued to the server by the XATMI-PM on receipt of a request for a conversational service. The server uses the *tpservice*( ) template to map the parameters and then dispatches the corresponding application routine.

The *tpservice*( ) template is mapped from the XATMI-CONNECT parameters as follows:

<table>
<tr><th colspan="2">tpservice( )</th><th>XATMI-CONNECT ind</th><th>Notes</th></tr>
<tr><td colspan="2">cd</td><td></td><td>Local processing.</td></tr>
<tr><td colspan="2">name</td><td>Service-Name</td><td>The Service-Name parameter is mapped from the service field of the XATMI-CONNECT-RI APDU received with the TP-DATA indication (see below).</td></tr>
<tr><td colspan="2">data, len</td><td>User-Data</td><td>The XATMI-PM uses the abstract syntax defined in Chapter 14, to decode the typed buffer from the the data field of the XATMI-CONNECT-RI APDU.</td></tr>
<tr><td rowspan="5">flags</td><td>TPCONV</td><td></td><td>This flag is set when this indication type is received by the server.</td></tr>
<tr><td>TPTRAN</td><td>Begin-Transaction = True</td><td>The service is invoked as part of the client's transaction.</td></tr>
<tr><td>TPRECVONLY</td><td>Grant-Control = False</td><td>The Client retains control of the dialogue.</td></tr>
<tr><td>TPSENDONLY</td><td>Grant-Control = True</td><td>The Service has been granted control of the dialogue.</td></tr>
<tr><td>TPNOREPLY</td><td></td><td>Not permitted for conversational Services.</td></tr>
</table>

13.14.2 Mapping from OSI TP

The XATMI-CONNECT indication parameters are generated as follows:

- The Service-Name parameter is mapped from the **service** field of the XATMI-CONNECT-RI APDU (see Chapter 14).
- The Begin-Transaction parameter is set to **True** if the Chained Transaction functional unit is selected or if the Unchained Transaction functional unit is selected and the Begin-Transaction parameter in the TP-BEGIN-DIALOGUE indication is set to **True**.
- The Grant-Control parameter value is set to **True** if MAP 29 is detected by the XATMI-PM; otherwise the value is set to **False** (MAP 30).
- The User-Data parameter is set if the **data** field of the XATMI-CONNECT-RI APDU is present. This field is decoded into a local typed buffer structure following the rules described in Chapter 14.

The different mappings are as follows:

MAP 29: This mapping is generated when the client requests a connection with a conversational service in **receive** mode (TPRECVONLY, see Section 13.7.1 on page 157)

1. Upon receiving a TP-BEGIN-DIALOGUE indication, the XATMI-PM starts an XATMI-CONNECT invocation procedure. This procedure performs validations against the local configuration information, and if any validation fails MAP 31 is then performed.

2. A TP-DEFERRED-END-DIALOGUE indication is expected by the procedure if the Begin-Transaction parameter of the TP-BEGIN-DIALOGUE indication is set to **True**. No action is taken by the XATMI-PM upon receiving this indication. Note that this indication may be received at any time until an XATMI-PREPARE-RI APDU is received.

3. An XATMI-CONNECT-RI APDU is then received. This APDU is decoded following the rules specified in Chapter 14. The Service-Name and User-Data parameters of the XATMI-CONNECT indication are set from the values of the **service** and **data** fields of the XATMI-CALL-RI APDU.

4. A TP-GRANT-CONTROL indication is then received. The XATMI-PM sets to **True** the Grant-Control parameter of the XATMI-CONNECT indication, terminates the XATMI-CONNECT invocation procedure, and issues this indication to the server. Upon receiving the XATMI-CONNECT indication, the server invokes the corresponding application routine, mapping the parameters as specified in Section 13.14.1 on page 167.

MAP 30: This mapping is generated when the client requests a connection to a conversational service in **send** mode (TPSENDONLY, see Section 13.7.1 on page 157)

1. Upon receiving a TP-BEGIN-DIALOGUE indication, the XATMI-PM starts an XATMI-CONNECT invocation procedure. This procedure performs validations against the local configuration information, and if any validation fails MAP 31 is then performed. If the Confirmation parameter of the TP-BEGIN-DIALOGUE indication is to **Always**, this mapping is applied; otherwise MAP 29 is applied.

2. A TP-DEFERRED-END-DIALOGUE indication is expected by the procedure if the Begin-Transaction parameter of the TP-BEGIN-DIALOGUE indication is set to **True**. No action is taken by the XATMI-PM upon receiving this indication. Note that this indication may be received at any time up until an XATMI-PREPARE-RI APDU is received.

3. An XATMI-CONNECT-RI APDU is then received. This APDU is decoded following the rules specified in Chapter 14. The Service-Name and User-Data parameters of the XATMI-CONNECT indication are set from the values of the **service** and **data** fields of the XATMI-CALL-RI APDU.

4. After receiving the XATMI-CONNECT-RI APDU, the XATMI-PM issues a TP-BEGIN-DIALOGUE response accepting the dialogue.

5. The XATMI-PM then sets the Grant-Control parameter of the XATMI-CONNECT indication to **False**, and issues an XATMI-CONNECT indication to the server. The server then invokes the corresponding application routine.

MAP 31: This mapping is generated when the XATMI-CONNECT-invocation procedure fails a validation. The dialogue is rejected with a TP-BEGIN-DIALOGUE response (see Section 13.13.2 on page 164, MAP 28).

13.15 XATMI-REPLY indication

13.15.1 Mapping to tpcall( ), tpgetrply( ), and tprecv( )

An XATMI-REPLY indication is mapped to *tpcall*( ), *tpgetrply*( ), or *tprecv*( ).

The XATMI-REPLY indication parameters are mapped as follows:

XATMI-REPLY ind	Mapping
User-Code	This value contains the return code (*rcode*) generated by the application with the call to *tpreturn*( ). This value is mapped to the *tpurcode* global variable.
User-Data	The XATMI-PM sets this parameter from the value of the **data** field of the XATMI-REPLY-RI APDU. This parameter is mapped as follows: *tpcall*( ): to *odata* and *olen*, *tpgetrply*( ): to *data* and *len*, *tprecv*( ): to *data* and *len*. Also, the *rvent* variable is set to TPEV_SVCSUCC.

13.15.2 Mapping from OSI TP

The XATMI-REPLY indication parameters are mapped from an XATMI-REPLY-RI APDU. The User-Data parameter is mapped from the **data** field of the XATMI-REPLY-RI APDU.

The following is a description of the different mappings:

MAP 3: This mapping is generated when the service request is invoked within a global transaction.

1. Upon receiving an XATMI-REPLY-RI APDU, the XATMI-PM starts an XATMI-REPLY-invocation procedure to decode the APDU according to the rules described in Chapter 14. The value of the User-Code parameter is set to the value of the **user-code** field of this APDU. The value of the User-Data parameter contains a typed buffer built from the **data** field of this APDU (if it exists), otherwise it is set to null.
2. A TP-GRANT-CONTROL indication is then received. When this indication is received, the XATMI-REPLY invocation procedure completes and the XATMI-PM issues an XATMI-REPLY indication to the client.

MAP 4: This mapping is generated when the application expects a reply to a service request issued outside any global transaction.

1. Upon receiving an XATMI-REPLY-RI APDU, the XATMI-PM starts an XATMI-REPLY invocation procedure to decode the APDU according to the rules described in Chapter 14. The value of the User-Code parameter is set to the value of the **user-code** field of this APDU. The value of the User-Data parameter contains a typed buffer built from the **data** field of this APDU (if it exists), otherwise it is set to null.
2. When the TP-END-DIALOGUE indication is received, the XATMI-REPLY-indication procedure completes and the XATMI-PM issues an XATMI-REPLY indication to the client.

13.16 XATMI-FAILURE indication

13.16.1 Mapping to tpcall( ), tpgetrply( ), tpsend( ), and tprecv( )

An XATMI-FAILURE indication is mapped to *tpcall*( ), *tpgetrply*( ), *tpsend*( ) or *tprecv*( ).

The XATMI-FAILURE failure parameters are mapped as follows:

XATMI-FAILURE ind	**Mapping**
Diagnostic ={	This parameter is mapped to a return code from *tpcall*( ) and *tpgetrply*( ), or to the *revent* parameter of *tpsend*( ) and *tprecv*( ).
Application-Service-Failure	Mapped to TPESVCFAIL or TPEV_SVCFAIL.
Recipient-XATMI-SU-Failure Rejected-XATMI-Provider Permanent-Failure Transient-Failure Protocol-Error Recipient-TPSU-title-unknown Recipient-TPSU-title-required TPSU-not-available(permanent) TPSU-not-available(transient) Functional-Unit-combination-not-supported Reason-not-specified }	All other Diagnostic values are mapped to TPESVCERR or to TPEV_SVCERR.
User-Code	This value contains the return code (*rcode*) generated by the application with the call with *tpreturn*( ). This value is mapped to the *tpurcode* global variable.
User-Data	This parameter contains a typed buffer that is constructed from the value of the **data** field of the XATMI-FAILURE-RI APDU. This parameter is mapped as follows: *tpcall*( ): to *odata* and *olen*, *tpgetrply*( ): to *data* and *len*, *tprecv*( ): to *data* and *len*.

Note that when the Diagnostic parameter is set to a value other than **Application-Service-Failure**, the User-Code and User-Data parameters are set to null.

13.16.2 Mapping from OSI TP

An XATMI-FAILURE indication is mapped from OSI TP as follows:

MAP 5: This mapping is generated when the remote application service ended with a call to *tpreturn*( ) with the *rcode* value set to TPFAIL or when a Transaction Processing Service User failure occurs.

When the XATMI-PM receives a TP-U-ABORT indication, it starts an XATMI-FAILURE indication procedure that maps the TP-U-ABORT parameters as follows:

TP-U-ABORT	ind	XATMI-FAILURE ind Procedure
Rollback	M	If the transaction associated with the OSI TP dialogue is being rolled back, this parameter is set to **True**. The XATMI-PM marks the global transaction as ROLLBACK-IN-PROGRESS. This is so that the XATMI-PM does not generate improper protocol when the application eventually issues *tx_rollback*( ). If the parameter is set to **False**, no action is taken by the XATMI-PM.
User-Data	U	If User-Data is not present, the indication is treated as a Recipient-XATMI-SU-Failure. If User-Data is present, this parameter contains an XATMI-FAILURE-RI APDU according to the rules specified in Chapter 14. This APDU contains a **diagnostic** field and an optional **reply** field. If the **diagnostic** field is set to Application-Service-Failure, then the **reply** field must be present and the **diagnostic**, **user-code**, and **data** fields of the APDU are mapped to the corresponding fields of the XATMI-FAILURE indication. If the **diagnostic** field is set to any other value, the **reply** field will not be present.

MAP 6: This mapping is generated when there is a network or a TP provider failure. The XATMI-PM uses this mapping when a TP-P-ABORT indication is received, and there is a pending reply from the remote application service.

The parameters of the TP-P-ABORT indication are mapped as follows:

TP-P-ABORT	**ind**	**XATMI-FAILURE ind Procedure**
Rollback	M	If the transaction associated with the OSI TP dialogue is being rolled back, this parameter is set to **True**. The XATMI-PM marks the global transaction as ROLLBACK-IN-PROGRESS. This is so that the XATMI-PM does not generate improper protocol when the application eventually issues *tx_rollback*( ). If the parameter is set to **False**, no action is taken by the XATMI-PM.
Diagnostic = {	M	This parameter is mapped to the Diagnostic parameter of the XATMI-FAILURE indication as follows:
Permanent-failure		Same value
Transient-failure		Same value
Protocol-error		Same value
Begin-transaction-reject		Cannot happen (see note below)
End-dialogue-collision		Cannot happen (see note below)
Begin-transaction-end-dialogue-collision		Cannot happen (see note below).
}		**Note:** Several Diagnostic values cannot occur because of the OSI TP mapping defined by the XATMI-ASE and the use of OSI TP profiles ATP21 and ATP31.

MAP 7: This mapping is generated when the remote XATMI-PM or the remote TP-PM rejects a dialogue establishment. The XATMI-PM uses this mapping when a TP-BEGIN-DIALOGUE confirm is received, and there is a pending reply from the remote application service (that is, the XATMI-PM is not within an XATMI-CONNECT request procedure).

TP-BEGIN-DIALOGUE	cnf	XATMI-FAILURE ind Procedure
Rollback	M	If the transaction associated with the OSI TP dialogue is being rolled back, this parameter is set to **True**. The XATMI-PM marks the global transaction as ROLLBACK-IN-PROGRESS. If the parameter is set to **False**, no action is taken by the XATMI-PM.

Result = {
Accepted
 	
Rejected(user)
 	
Rejected(Provider)
 	
}
	M	
		Cannot happen — XATMI-ASE uses unconfirmed dialogues.
		Dialogue rejected by the remote XATMI-ASE.
		Dialogue rejected by the remote TP-PM.
		Note: A rejected(user) value is mapped to value Rejected-XATMI-Provider on the Diagnostic parameter of the XATMI-FAILURE indication. A rejected(provider) is mapped as indicated below.

Functional Units	C	An XATMI-PM provider may record this value somewhere.

Diagnostic = {
 	
Recipient-unknown
Recipient-TPSU-title-unknown
Recipient-TPSU-title-required
TPSU-not-available (permanent)
TPSU-not-available (transient)
FU-combination-not-supported
Reason-not-specified
}
	C
	
	(=)
	(=)
	(=)
	(=)
	(=)
	(=)
	(=)
		Mapped to Diagnostic on the XATMI-FAILURE indication with the same value.

User-Data	U	Not used.

Notes:

1. The Diagnostic parameter of an XATMI-FAILURE indication may be set from the Diagnostic parameter of a TP-P-ABORT indication, from the Diagnostic parameter of a TP-BEGIN-DIALOGUE confirm, or from the **diagnostic** field value of an XATMI-FAILURE-RI APDU.

2. If an XATMI-FAILURE indication is received by an XATMI-SUI that is within a global transaction, its transaction is being rolled back by the XATMI-PM. The XATMI-SUI must eventually roll back its local data, usually triggered by *tx_rollback*( ), and then issue an XATMI-DONE request (see Section 13.20.3 on page 179) to allow the XATMI-PM to complete the rollback.

3. If the XATMI-FAILURE indication initiates rollback of the current transaction, any TP-P-ABORT indication or TP-U-ABORT indication primitives received by the XATMI-PM between the XATMI-FAILURE indication and the subsequent XATMI-DONE request are silently discarded by the XATMI-PM. TP-U-ABORT requests for any remaining dialogues take place when the XATMI-DONE request is issued.

13.17 XATMI-CANCEL indication

13.17.1 Mapping to the XATMI Interface

An XATMI_CANCEL indication is not mapped to the XATMI Interface. When a server receives this indication, it should not issue an XATMI-REPLY request.

An XATMI-CANCEL indication applies only to request/response services invoked outside the client's global transaction.

13.17.2 Mapping from OSI TP

An XATMI-CANCEL indication is issued by the server XATMI-PM according to the following mappings:

MAP 35: This mapping is only generated when the client issues *tpcancel*( ). The XATMI-PM maps a TP-U-ABORT indication to an XATMI-CANCEL. The User-Data and the Rollback parameters of the TP-U-ABORT indication are ignored.

MAP 36: This mapping is generated when there is a network or a TPPM failure. The XATMI-PM maps the TP-P-ABORT indication to an XATMI-CANCEL. The Diagnostic and Rollback parameters are ignored.

13.18 XATMI-DISCON indication

13.18.1 Mapping to tpsend( ) and tprecv( )

An XATMI-DISCON indication can only be produced at the server XATMI-PM when the client issues *tpdiscon*( ) or when a communication failure occurs in a connection with a conversational service.

An XATMI-DISCON indication is translated to the event TPEV_DISCONIMM that is returned on the *revent* variable of *tpsend*( ) or *tprecv*( ).

13.18.2 Mapping from OSI TP

An XATMI-DISCON indication is issued by a server XATMI-PM according to the following mappings:

MAP 37: The XATMI-PM issues an XATMI-DISCON indication when a TP-U-ABORT indication is received in a dialogue associated with an active conversational service. If the Rollback parameter is set to **True**, the transaction is marked as ROLLBACK-IN-PROGRESS. In this case, the User-Data parameter of the TP-U-ABORT indication is not mapped.

MAP 38: The XATMI-PM issues an XATMI-DISCON indication when a TP-P-ABORT indication is received in a dialogue associated with an active conversational service. If the Rollback parameter is set to **True**, the transaction branch is marked as ROLLBACK-IN-PROGRESS. The Diagnostic parameter of the TP-P-ABORT indication is not mapped.

Notes:

1. If an XATMI-DISCON indication is received by an XATMI-SUI that is within a global transaction, its transaction is being rolled back by the XATMI-PM. The XATMI-SUI must eventually roll back its local data, usually triggered by *tx_rollback*( ), and then issue an XATMI-DONE request (see Section 13.20.3 on page 179) to allow the XATMI-PM to complete the rollback.
2. If the XATMI-DISCON indication initiates rollback of the current transaction, any TP-P-ABORT indication or TP-U-ABORT indication primitives received by the XATMI-PM between the XATMI-DISCON indication and the subsequent XATMI-DONE request are silently discarded by the XATMI-PM. TP-U-ABORT requests for any remaining dialogues take place when the XATMI-DONE request is issued.

13.19 XATMI-DATA indication

13.19.1 Mapping to tprecv( )

An XATMI-DATA indication is mapped to *tprecv*( ).

The XATMI-DATA indication parameters are mapped as follows:

XATMI-DATA ind	Mapping
Grant-Control	This parameter is set to **True** when the sender grants control of the conversation. Otherwise, this parameter is set to **False**.
User-Data	The XATMI-PM decodes the *data* field of the XATMI-DATA-RI or XATMI-DATA-GRANT-CONTROL-RI APDU, and converts it into a local typed buffer. This parameter is then mapped to the *data* and *len* parameters of the *tprecv*( ) primitive.

13.19.2 Mapping from OSI TP

An XATMI-DATA indication is generated by the XATMI-PM according to the following mappings:

MAP 13: This mapping is generated when the XATMI-PM instance for a particular conversation is in **Receive** mode and the following indications are received:

1. Upon receiving an XATMI-DATA-RI or XATMI-DATA-GRANT-CONTROL-RI APDU, the XATMI-PM decodes the APDU according to the rules described in Chapter 14. If the **data** field is present, the XATMI-PM constructs a typed buffer and returns it in the User-Data parameter of the XATMI-DATA indication.

2. If an XATMI-DATA-RI APDU was received, the XATMI-PM sets the Grant-Control parameter to **False** and issues an XATMI-DATA indication to the receiver (the client or server). Alternatively if an XATMI-DATA-GRANT-CONTROL-RI APDU was received the XATMI-PM waits for the TP-GRANT-CONTROL indication, sets the Grant-Control parameter to **True** and issues the XATMI-DATA indication.

MAP 39: This mapping is the same as MAP 13 but on the server XATMI-PM.

13.20 Mapping Transaction Services

The mappings for the XATMI-ASE transaction services are generated when the application issues a *tx_commit*( ) or a *tx_rollback*( ) primitive (see Section 12.3 on page 123).

The behaviour of the XATMI-ASE transaction services is directly inferred from the corresponding mapped OSI TP services (see below).

13.20.1 XATMI-PREPARE request

Mapping from the TX Interface

The XATMI-SUI maps a *tx_commit*( ) to an XATMI-PREPARE request. There are no parameters to be mapped.

A server XATMI-SUI may also issue an XATMI-PREPARE request when it behaves as a client (that is, the application issued service requests or established conversations with other services). This request can only be issued when the server receives an XATMI-PREPARE indication from its superior.

Mapping to OSI TP

The mapping to OSI TP is as follows:

MAP 14: When an XATMI-PREPARE request is issued, the XATMI-MACF generates a TP-PREPARE-ALL request (see Section 13.1 on page 145). This service request is global in that it affects all subordinate transaction branches, and allows a true separation of the two phases of the commitment of a global transaction. The TP-PREPARE-ALL request works as described in the referenced **XAP-TP** specification (see Section 13.1 on page 145).

13.20.2 XATMI-COMMIT request

Mapping from the TX Interface

An XATMI-COMMIT request is not mapped directly from the XATMI interface or the TX interface.

Mapping to OSI TP

An XATMI-COMMIT request is issued by the XATMI-SUI when local data is in the **Ready** state, and one of the following conditions is met:

- The XATMI-PM has issued an XATMI-READY indication.
- The XATMI-PM has issued an XATMI-PREPARE indication, and there are no subordinate dialogues.

The mapping to OSI TP is as follows:

MAP 16: An XATMI-COMMIT request is translated into the abstract service TP-COMMIT-ALL request (see Section 13.1 on page 145). This service starts the second phase of commitment. The combination of a TP-PREPARE-ALL request and a TP-COMMIT-ALL request as specified in Section 13.1 on page 145 is the equivalent of the OSI TP abstract service TP-COMMIT as defined in the OSI TP Service standard. Upon receiving a TP-COMMIT-ALL request, the OSI TPPM considers the transaction committed.

MAP 41: This mapping is the same as MAP 16 but it is generated by the server XATMI-PM.

13.20.3 XATMI-DONE request

Mapping from the TX Interface

An XATMI-DONE request is not mapped directly from the TX interface.

Mapping to OSI TP

An XATMI-DONE request is issued when the client (or the server) has released its data into the final state.

An XATMI-DONE maps to OSI TP as follows:

MAP 18: This mapping is generated when a client has committed its local data (that is, the XATMI-DONE request has been issued after an XATMI-COMMIT indication). The XATMI-PM maps the XATMI-DONE request to a TP-DONE request — no parameters are mapped in this case.

Note: Any TP-P-ABORT indication primitives received by the XATMI-PM between the XATMI-DONE request and the subsequent TP-COMMIT-COMPLETE indication are processed by the XATMI-PM which issues a TP-DONE request (with no parameters) to acknowledge each. (All the dialogues for the transaction node will, in any case, cease to exist upon commit completion.)

MAP 19: This mapping is generated after a client has rolled back its local data (that is, the XATMI-DONE request is issued after a rollback initiating primitive has been issued). The XATMI-PM maps the XATMI-DONE request to OSI TP as follows:

1. A TP-U-ABORT request is issued for each dialogue that has not yet been aborted. The User-Data parameter is not mapped.
2. A TP-DONE request is issued with no parameters.

MAP 44: This mapping is generated when a server has committed its local data (that is, the XATMI-DONE request has been issued after an XATMI-COMMIT indication). The XATMI-PM maps the XATMI-DONE request to a TP-DONE request; if the Heuristic-Report parameter is present on the XATMI-DONE request, this is mapped to the Heuristic-Report parameter of the TP-DONE request. The possible values of the XATMI-DONE request Heuristic-Report parameter are specified in Section 12.4.11 on page 138.

Note: Any TP-P-ABORT indication primitives received by the XATMI-PM between the XATMI-DONE request and the subsequent TP-COMMIT-COMPLETE indication are processed by the XATMI-PM which issues a TP-DONE request (with no parameters) to acknowledge each. (All the dialogues for the transaction node will, in any case, cease to exist upon commit completion.)

MAP 45: This mapping is generated after a server has rolled back its local data (that is, the XATMI-DONE request is issued after a rollback initiating primitive has been issued). The XATMI-PM maps the XATMI-DONE request to OSI TP as follows:

1. A TP-U-ABORT request is issued for each dialogue that has not yet been aborted. The User-Data parameter is not mapped.

2. A TP-DONE request is issued; if the Heuristic-Report parameter is present on the XATMI-DONE request, this is mapped to the Heuristic-Report parameter of the TP-DONE request. The possible values of the XATMI-DONE request Heuristic-Report parameter are specified in Section 12.4.11 on page 138.

13.20.4 XATMI-ROLLBACK request

Mapping from the TX Interface

An XATMI-ROLLBACK request is mapped from *tx_rollback*( ). There are no parameters to be mapped.

Mapping to OSI TP

An XATMI-ROLLBACK request is mapped to OSI TP as follows:

MAP 20: An XATMI-ROLLBACK request is mapped by the client XATMI-PM to a TP-ROLLBACK request. This mapping is generated when the client application issued *tx_rollback*( ).

MAP 46: An XATMI-ROLLBACK request is mapped by the server XATMI-PM to a TP-U-ABORT request. The User-Data parameter of the TP-U-ABORT service is not used. This mapping is generally used when the server XATMI-SUI fails to bring its local data to the **Ready** state.

Note: Any TP-P-ABORT indication or TP-U-ABORT indication primitives received by the XATMI-PM between the XATMI-ROLLBACK request and the subsequent XATMI-DONE request are silently discarded by the XATMI-PM. TP-U-ABORT requests for any remaining dialogues take place when the XATMI-DONE request is issued.

13.20.5 XATMI-PREPARE indication

Mapping to the TX Interface

An XATMI-PREPARE is not mapped directly to the TX interface.

Mapping from OSI TP

An XATMI-PREPARE indication is issued by the XATMI-PM as follows:

MAP 41: The XATMI issues an XATMI-PREPARE indication after receiving a TP-PREPARE indication. The Data-Permitted parameter is not mapped.

13.20.6 XATMI-READY indication

Mapping to the TX Interface

An XATMI-READY indication is not mapped directly to the TX interface.

Mapping from OSI TP

An XATMI-READY indication is issued by the XATMI-MACF when all subordinate dialogues are in the **Ready** state.

The mapping from OSI TP is the following:

MAP 15: The XATMI-PM (MACF) issues an XATMI-READY indication when it has received a TP-READY-ALL indication (see Section 13.1 on page 145). A TP-READY-ALL indication works as specified in the **XAP-TP** specification, and it is issued by the OSI TPPM to indicate that the transaction has been brought to a **Ready** state (this node and all its subordinate branches are **Ready**).

13.20.7 XATMI-COMMIT indication

Mapping to the TX Interface

An XATMI-COMMIT is not mapped directly to the TX interface.

Mapping from OSI TP

An XATMI-COMMIT indication is mapped from OSI-TP as follows:

MAP 17: An XATMI-COMMIT indication is issued when the client XATMI-PM receives a TP-COMMIT indication. This indication is issued to allow the XATMI-SUI release its bound data in the final state.

MAP 43: This mapping is the same as MAP 17 but it is generated at the server XATMI-PM.

Note: Any TP-P-ABORT indication primitives received by the XATMI-PM between the XATMI-COMMIT indication and the subsequent XATMI-DONE request are silently discarded by the XATMI-PM. TP-U-ABORT requests for any will in any case cease to exist at the completion of commitment as a result of the outstanding TP-DEFERRED-END-DIALOGUE requests. (Under these circumstances it is not possible to receive a TP-U-ABORT indication.)

13.20.8 XATMI-ROLLBACK indication

Mapping to the TX Interface

An XATMI-ROLLBACK indication maps to a *tx_commit*( ) return code. It also maps to the TPEV_DISCONIMM event in *tprecv*( ) or *tpsend*( ).

Mapping from OSI TP

An XATMI-ROLLBACK indication is mapped from OSI TP as follows:

MAP 21: An XATMI-ROLLBACK indication is issued when a TP-P-ABORT indication is received, the Rollback parameter of this indication is set to **True**, and the transaction is in the **Prepare** phase (**Prepared-issued** or **Ready** state). This mapping is generated when there is a communication failure or a TPPM failure during the **Prepare** phase of the transaction.

MAP 22: An XATMI-ROLLBACK indication is issued when a TP-U-ABORT indication is received, the Rollback parameter of this indication is set to **True**, and the transaction is in the **Prepare** phase (**Prepared-issued** or **Ready** state). This mapping is usually produced when the server XATMI-PM did not receive a TP-DEFERRED-END-DIALOGUE indication before a TP-PREPARE indication or when the server XATMI-SUI fails to bring its local data to the **Ready** state (see MAP 46).

MAP 47: An XATMI-ROLLBACK indication is issued when a TP-ROLLBACK indication is received by the server XATMI-PM. This indication is the result of *tx-rollback*( ) issued by the client application.

MAP 48: This is the same as MAP 21 but it is generated by the server XATMI-PM.

Note: Any TP-P-ABORT indication or TP-U-ABORT indication primitives received by the XATMI-PM between the XATMI-ROLLBACK indication and the subsequent XATMI-DONE request are silently discarded by the XATMI-PM. TP-U-ABORT requests for any remaining dialogues take place when the XATMI-DONE request is issued.

13.20.9 XATMI-COMPLETE indication

Mapping to the TX Interface

tx_commit( ) or *tx_rollback*( ) formally completes when an XATMI-COMPLETE indication is received.

Mapping from OSI TP

An XATMI-COMPLETE indication is mapped from OSI-TP as follows:

MAP 23: An XATMI-COMPLETE indication is issued when the client XATMI-PM receives a TP-COMMIT-COMPLETE indication. The transaction commitment has been completed by the XATMI-PM.

MAP 24: An XATMI-COMPLETE indication is issued when the client XATMI-PM receives a TP-ROLLBACK-COMPLETE indication. The transaction rollback has been completed by the XATMI-PM.

MAP 49: This mapping is the same as MAP 23 but it is generated by the server XATMI-PM.

MAP 50: This mapping is the same as MAP 24 but it is generated by the server XATMI-PM.

13.20.10 XATMI-HEURISTIC indication

Mapping to the TX Interface

An XATMI-HEURISTIC indication maps to a return code on *tx_commit*( ) or *tx_rollback*( ).

Mapping from OSI TP

An XATMI-HEURISTIC indication is mapped as follows:

MAP 25: An XATMI-HEURISTIC indication is issued when the client XATMI-PM receives a TP-HEURISTIC-REPORT indication on any dialogue associated with the global transaction. The value of the Heuristic-Report parameter is mapped to the value of the Diagnostic parameter of the XATMI-HEURISTIC indication (that is, the same value is used).

13.21 Mapping to the XATMI Interface Return Codes

The mapping of return codes from the XATMI-ASE service indications is as follows:

Table 13-5 XATMI-ASE Return Code Mappings

Return Code	Mapping from XATMI-ASE
TPEINVAL	Returned by *tpacall*( ), *tpcall*( ), *tpgetrply*( ), *tpconnect*( ), *tpsend*( ) and *tprecv*( ); results from local processing by the XATMI Provider.
TPENOENT	Returned by *tpcall*( ), *tpacall*( ) and *tpconnect*( ); may indicate an error in the local configuration information or result from an XATMI-FAILURE indication.
TPEITYPE	Returned by *tpacall*( ), *tpcall*( ) and *tpconnect*( ); results when the type and subtype of the input buffer do not match those supported by the XATMI-ASE.
TPEOTYPE	Returned by *tpcall*( ), *tpgetrply*( ) and *tpconnect*( ); results when the type and subtype of the received reply buffer from the service cannot be mapped into the receiver's typed buffer.
TPETRAN	Returned by *tpcall*( ), *tpacall*( ), *tpconnect*( ) and *tpcancel*( ); may indicate an error in the local configuration information or result from an XATMI-FAILURE indication.
TPETIME	Returned by tpacall, tpcall, tpgetrply, tpconnect, tpdiscon, tpsend, tprecv; results from local processing.
TPESVCFAIL	Returned by *tpcall*( ) and *tpgetrply*( ); results from an XATMI-FAILURE indication (determined by the Diagnostic parameter, see Section 13.16 on page 170).
TPESVCERR	Returned by *tpcall*( ) and *tpgetrply*( ); results from an XATMI-FAILURE indication (determined by the Diagnostic parameter, see Section 13.16 on page 170).
TPEBLOCK	Returned by *tpacall*( ), *tpcall*( ), *tpgetrply*( ), *tpconnect*( ), *tpsend*( ) and *tprecv*( ); results from local processing.
TPGOTSIG	Returned by *tpacall*( ), *tpcall*( ), *tpgetrply*( ), *tpconnect*( ), *tpsend*( ) and *tprecv*( ); results from local processing.
TPEPROTO	Returned by all functions; results when a function is called in an invalid state.
TPESYSTEM	Returned by all functions; results from local processing.
TPEOS	Returned by all functions; results from local processing.
TPELIMIT	Returned by *tpgetrply*( ) and *tpconnect;*( ) results from local processing.
TPEBADDESC	Returned by *tpgetrply*( ), *tpcancel*( ), *tpdiscon*( ), *tpsend*( ) and *tprecv*( ); results from local processing.

Return Code	**Mapping from XATMI-ASE**
TPEEVENT	One of the following events has occurred: TPEV_DISCONIMM: returned by *tpsend*( ) and *tprecv*( ); results from an XATMI-FAILURE indication, an XATMI-DISCON indication or an XATMI-ROLLBACK indication. TPEV_SVCFAIL: returned by *tpsend*( ) and *tprecv*( ); results from an XATMI-FAILURE indication. TPEV_SVCERR: returned by *tpsend*( ) and *tprecv*( ); results from an XATMI-FAILURE indication. TPEV_SENDONLY: returned by *tprecv*( ); results from an XATMI-DATA indication. TPEV_SVCSUCC: returned by *tprecv*( ); service has finished (an XATMI-REPLY indication was received).

Chapter 14

Structure and Encoding of XATMI-ASE APDUs

This chapter defines the structure and encoding of the XATMI-ASE Application Protocol Data Units (APDUs) as well as the mappings of X/Open's XATMI Buffer Types to these APDUs.

14.1 Abstract Syntax

The abstract syntax of each APDU is specified using ASN.1 (see the referenced ASN.1 standard).

```
-- XATMI-ASE APDUs
XATMI-APDUs

-- An OBJECT IDENTIFIER must be registered by X/Open
{iso(1) national-member-body(2) bsi(826) disc(0) xopen(1050)
        xatmi(4) apdus-abstract-syntax(1) version1(0)}

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS
-- all definitions --

-- top level APDU CHOICE

XATMI-APDU ::=  CHOICE
{
        xatmi-call-ri               [1] XATMI-CALL-RI,
        xatmi-reply-ri              [2] XATMI-REPLY-RI,
        xatmi-failure-ri            [3] XATMI-FAILURE-RI,
        xatmi-connect-ri            [4] XATMI-CONNECT-RI,
        xatmi-data-ri               [5] XATMI-DATA-RI,
        xatmi-data-grant-control-ri [6] XATMI-DATA-GRANT-CONTROL-RI

}

-- individual APDU definitions

XATMI-CALL-RI ::=  SEQUENCE
{
        service                     [1] VisibleString,
        data                        [2] XATMI-typed-buffer OPTIONAL
}

XATMI-REPLY-RI ::=  SEQUENCE
{
        user-code                   [1] INTEGER,
        data                        [2] XATMI-typed-buffer OPTIONAL
}
```

```
XATMI-FAILURE-RI ::= SEQUENCE
{
        diagnostic                  [1] ENUMERATED
                                       {recipient-xatmi-su-failure      (10),
                                        application-service-failure     (11)
                                       },
        reply                       [2] XATMI-REPLY-RI OPTIONAL
}

XATMI-CONNECT-RI ::= XATMI-CALL-RI

XATMI-DATA-RI ::= SEQUENCE
{
        data                        [1] XATMI-typed-buffer OPTIONAL
}

XATMI-DATA-GRANT-CONTROL-RI ::= SEQUENCE
{
        data                        [1] XATMI-typed-buffer OPTIONAL
}

-- supporting type definitions --

XATMI-typed-buffer ::= SEQUENCE
{
        type                        [1] VisibleString,
        subtype                     [2] VisibleString OPTIONAL,
        data                        [3] XATMI-buffer-types
}

XATMI-buffer-types ::= CHOICE
{
        X-octet                     [1] OCTET STRING,
        X-common                    [2] SEQUENCE OF X-common,
        X-c-type                    [3] SEQUENCE OF X-c-type,
        X-u-defined                 [4] SEQUENCE OF ANY
}

X-common ::= CHOICE
{
        short                       [1] INTEGER,
        short-n                     [2] SEQUENCE OF INTEGER,
        long                        [3] INTEGER,
        long-n                      [4] SEQUENCE OF INTEGER,
        char                        [5] OCTET STRING,
        char-n                      [6] OCTET STRING,
        char-translate-n            [7] T61String,
        char-translate              [8] T61String
}
```

```
X-c-type ::= CHOICE
{
        short                       [1]  INTEGER,
        short-n                     [2]  SEQUENCE OF INTEGER,
        integer                     [3]  INTEGER,
        integer-n                   [4]  SEQUENCE OF INTEGER,
        long                        [5]  INTEGER,
        long-n                      [6]  SEQUENCE OF INTEGER,
        char                        [7]  OCTET STRING,
        char-n                      [8]  OCTET STRING,
        char-translate-n            [9]  T61String,
        float                       [10] REAL,
        float-n                     [11] SEQUENCE OF REAL,
        double                      [12] REAL,
        double-n                    [13] SEQUENCE OF REAL,
        char-translate              [17] T61String,
        char-n-n                    [18] SEQUENCE OF OCTET STRING,
        char-translate-n-n          [19] SEQUENCE OF T61String,
        char-string                 [20] OCTET STRING,
        char-string-n               [21] SEQUENCE OF OCTET STRING,
        char-string-translate       [22] T61String,
        char-string-translate-n     [23] SEQUENCE OF T61String
}

END -- of XATMI-ASE definitions
```

14.2 Mapping X/Open XATMI Buffer Types

The X/Open XATMI buffer types, X_OCTET, X_COMMON and X_C_TYPE, are valid settings for the **type** element of the XATMI-typed-buffer SEQUENCE defined in the previous section.

The following table shows how the X/Open XATMI Buffer Types map to ASN.1.

Buffer Type	ASN.1 Type
X_OCTET	OCTET STRING
X_COMMON	SEQUENCE
X_C_TYPE	SEQUENCE

Table 14-1 Mapping of XATMI Buffer Types to ASN.1

The following table summarises the mapping of the elements of the above X/Open Buffer Types to ASN.1. Mappings for both C and COBOL data types are shown.

Buffer Type	C Type	COBOL Type	ASN.1 Type
X_OCTET	**char[*n*]**	**PIC X(*n*)**	[1] OCTET STRING
X_COMMON	**short**	**PIC S9(4) COMP-5**	[1] INTEGER
	short[*n*]	**PIC S9(4) COMP-5 OCCURS *n* TIMES**	[2] SEQUENCE OF INTEGER
	long	**PIC S9(9) COMP-5**	[3] INTEGER
	long[*n*]	**PIC S9(9) COMP-5 OCCURS *n* TIMES**	[4] SEQUENCE OF INTEGER
	char	**PIC X**	[5] OCTET STRING
	char	**PIC X**	[8] T61String
	char[*n*]	**PIC X(*n*)**	[6] OCTET STRING
	char[*n*]	**PIC X(*n*)**	[7] T61String
X_C_TYPE	**short**	N/A	[1] INTEGER
	short[*n*]	N/A	[2] SEQUENCE OF INTEGER
	integer	N/A	[3] INTEGER
	integer[*n*]	N/A	[4] SEQUENCE OF INTEGER
	long	N/A	[5] INTEGER
	long[*n*]	N/A	[6] SEQUENCE OF INTEGER
	char	N/A	[7] OCTET STRING
	char	N/A	[17] T61String
	char[*n*]	N/A	[8] OCTET STRING
	char[*n*]	N/A	[9] T61String
	char[*m*][*n*]	N/A	[18] SEQUENCE OF OCTET STRING
	char[*m*][*n*]	N/A	[19] SEQUENCE OF T61String
	float	N/A	[10] REAL
	float[*n*]	N/A	[11] SEQUENCE OF REAL
	double	N/A	[12] REAL
	double[*n*]	N/A	[13] SEQUENCE OF REAL
	char[*n*] null-terminated	N/A	[20] OCTET STRING
	char[*m*][*n*] null-terminated	N/A	[21] SEQUENCE OF OCTET STRING
	char[*n*] null-terminated	N/A	[22] T61String
	char[*m*][*n*] null-terminated	N/A	[23] SEQUENCE OF T61String

Table 14-2 Mapping of XATMI Buffer Type Elements to ASN.1

In the above table the notation **type[*n*]** for the C-language types refers to an unnamed array of *n* elements of type **type**. The notation **type[*m*][*n*]** refers to a two-dimensional array of type **type**. For the C types above denoted as null-terminated, the null terminator, or terminators in the case of two-dimensional arrays, is not transmitted in APDUs; rather, it is used locally to distinguish between ASN.1 types. The ASN.1 tags are shown in brackets preceding a data type's ASN.1 type.

When the XATMI CRM performs transparent encoding and decoding of character data, it maps the characters to either T.61 or PrintableString.

When mapping to or from PrintableString, an implementation must support the full set of PrintableString characters as defined by ASN.1 (a subset of the US ASCII printable characters).

When mapping to or from T.61, an implementation must support at least the following characters:

- T.61 primary control characters:

  BS position 0 / 8
  LF position 0 / 10
  FF position 0 / 12
  CR position 0 / 13

- T.61 Space, position 2 / 0

- T.61 primary graphic characters, positions 2 / 1 – 7 / 14, the US ASCII printable characters.

For this set of characters, interoperation is assured among all implementations of XATMI.

For the additional character sets and character extensions and escape sequences that can occur in T.61, interoperation may or may not be possible. This depends on which character sets are supported in the sending and receiving implementations.

X/Open CAE Specification

Part 3:

XATMI Communication API Appendices

X/Open Company Ltd.

Appendix A

C Programming Examples

This appendix contains examples that highlight the use of the XATMI functions; they are not meant to convey complete and correct programs.

A.1 Example 1

The following example shows the sequence of calls made by a Client Application and a Server Application that illustrates the use of the request/response Service Paradigm in the C programming language.

Client AP

```
DATA_BUFFER *dptr;      /* DATA_BUFFER is a typed buffer of type */
DATA_BUFFER *cptr;      /* X_C_TYPE and subtype dc_buf.  The structure */
long dlen, clen;        /* contains a character array named input and an */
int cd;                 /* integer named output. */

/* allocate typed buffers */
dptr = (DATA_BUFFER *) tpalloc("X_C_TYPE", "dc_buf", 0);
cptr = (DATA_BUFFER *) tpalloc("X_C_TYPE", "dc_buf", 0);

/* populate typed buffers with input data */
strcpy(dptr->input, "debit account 123 by 50");
strcpy(cptr->input, "credit account 456 by 50");

tx_begin();             /* start global transaction */

/* issue asynchronous request to DEBIT, while it is processing... */
cd = tpacall("DEBIT", (char *) dptr, 0, TPSIGRSTRT);

/* ...issue synchronous request to CREDIT */
tpcall("CREDIT", (char *) cptr, 0, (char **) &cptr, &clen, TPSIGRSTRT);

/* retrieve DEBIT's reply */
tpgetrply(&cd, (char **) &dptr, &dlen, TPSIGRSTRT);

if (dptr->output == OK && cptr->output == OK)
        tx_commit();    /* commit global transaction */
else
        tx_rollback();  /* rollback global transaction */
```

Service AP

```
/* this routine is used for DEBIT and CREDIT */
debit_credit_svc(TPSVCINFO *svcinfo)
{
        DATA_BUFFER *dc_ptr;
        int rval;

        /* extract request typed buffer */
        dc_ptr = (DATA_BUFFER *) svcinfo->data;

        /*
         * Depending on service name used to invoke this
         * routine, perform either debit or credit work.
         */
        if (!strcmp(svcinfo->name, "DEBIT")) {
                /*
                 * Parse input data and perform debit
                 * as part of global transaction.
                 */
        } else {
                /*
                 * Parse input data and perform credit
                 * as part of global transaction.
                 */
        }

        if (DBMS update successful) {
                rval = TPSUCCESS;
                dc_ptr->output = OK;
        } else {
                rval = TPFAIL;  /* global transaction will not commit */
                dc_ptr->output = NOT_OK;
        }
        /* send reply and return from service routine */
        tpreturn(rval, 0, (char *) dc_ptr, 0, 0);
}
```

A.2 Example 2

The following example shows the sequence of calls made by a Client Application and a Server Application that illustrates the use of the Conversational Service Paradigm in the C programming language.

Client AP

```
DATA_BUFFER *ptr;   /* DATA_BUFFER is a typed buffer of type */
long len, event;    /* X_C_TYPE and subtype inq_buf.  The structure */
int cd;             /* contains a character array named input and an */
                    /* array of integers named output. */

/* allocate typed buffer */
ptr = (DATA_BUFFER *) tpalloc("X_C_TYPE", "inq_buf", 0);

/* populate typed buffer with input data */
strcpy(ptr->input, "retrieve all accounts with balances less than 0");

tx_begin();             /* start global transaction */

/*connect to conversational service, send input data, & yield control*/
cd = tpconnect("INQUIRY", (char *) ptr, 0, TPRECVONLY|TPSIGRSTRT);

do {
        /* receive 10 account records at a time */
        tprecv(cd, (char **) &ptr, &len, TPSIGRSTRT, &event);
        /*
         * Format & display in AP-specific manner the accounts returned.
         */
} while (tperrno != TPEEVENT);

if (event == TPEV_SVCSUCC)
        tx_commit();    /* commit global transaction */
else
        tx_rollback();  /* rollback global transaction */
```

Service AP

```
/* this routine is used for INQUIRY */
inquiry_svc(svcinfo)
TPSVCINFO *svcinfo;
{
        DATA_BUFFER *ptr;
        long event;
        int rval;

        /* extract initial typed buffer sent as part of tpconnect() */
        ptr = (DATA_BUFFER *) svcinfo->data;

        /*
         * Parse input string, ptr->input, and retrieve records.
         * Return 10 records at a time to client.  Records are
         * placed in ptr->output, an array of account records.
         */

        do {
        /* gather from DBMS next 10 records into ptr->output array */
                tpsend(svcinfo->cd, (char *) ptr, 0, TPSIGRSTRT, &event);
        } while (more records exist);

        if (inquiry successful) {
                rval = TPSUCCESS;
        } else {
                rval = TPFAIL;  /* global transaction will not commit */
        }

        /* terminate service routine, send no data, and */
        /* terminate connection */
        tpreturn(rval, 0, NULL, 0, 0);
}
```

Appendix B

COBOL Programming Examples

This appendix contains examples that highlight the use of XATMI functions. These are not intended to convey complete and correct programs.

B.1 Example 1

The following example shows the sequence of calls made by a Client Application and a Server Application that illustrates the use of the request/response Service Paradigm in the COBOL programming language.

Client AP

```
 01 CREDIT-REQ
 COPY CREDIT.
*
 01 DEBIT-REQ
 COPY DEBIT.
*
* WK-AREA is used for replies.
*
 01 WK-AREA          PIC X(100).
*
 01 CREDIT-REP REDEFINES WK-AREA.
 COPY CREDIT.
*
 01 DEBIT-REP REDEFINES WK-AREA.
 COPY DEBIT.
*
* Start Global Transaction
*
 CALL "TXBEGIN".
*
* Set up TPTYPE-REC
*
 MOVE "X_COMMON" TO REC-TYPE.
 MOVE "dbuf" TO SUB-TYPE.
 MOVE LENGTH OF DEBIT-REQ TO LEN.
*
* Set up DEBIT-REQ
*
 MOVE "debit account 123 by 50" TO INPUT IN DEBIT-REQ.
*
* Set up the TPSVCDEF-REC
*
 MOVE LOW-VALUES TO TPSVCDEF-REC.
 MOVE "DEBIT" TO SERVICE-NAME.
*
 CALL "TPACALL" USING
        TPSVCDEF-REC TPTYPE-REC DEBIT-REQ TPSTATUS-REC.
```

```
*
* Set up TPTYPE-REC
*
 MOVE "X_COMMON" TO REC-TYPE.
 MOVE "cbuf" TO SUB-TYPE.
 MOVE LENGTH OF CREDIT-REQ TO LEN.
*
* Set up CREDIT-REQ
*
 MOVE "credit account 456 by 50" TO INPUT IN CREDIT-REQ.
*
* Set up the TPSVCDEF-REC
*
 MOVE LOW-VALUES TO TPSVCDEF-REC.
 MOVE "CREDIT" TO SERVICE-NAME.
*
 CALL "TPACALL" USING
        TPSVCDEF-REC TPTYPE-REC CREDIT-REQ TPSTATUS-REC.
*
* Set up TPTYPE-REC
*
 MOVE LENGTH OF WK-AREA TO LEN.
*
* Set up the TPSVCDEF-REC
*
 MOVE LOW-VALUES TO TPSVCDEF-REC.
 SET TPGETANY TO TRUE.
 CALL "TPGETRPLY" USING
        TPSVCDEF-REC TPTYPE-REC WK-AREA TPSTATUS-REC.
*
* Check SUB-TYPE to determine which record was received
*
 IF SUB-TYPE = "cbuf"
        MOVE OUTPUT IN CREDIT-REP TO OUTPUT IN CREDIT-REQ
 ELSE
        MOVE OUTPUT IN DEBIT-REP TO OUTPUT IN DEBIT-REQ.
*
* Set up TPTYPE-REC for second reply
*
 MOVE LENGTH OF WK-AREA TO LEN.
*
* Set up the TPSVCDEF-REC
*
 MOVE LOW-VALUES TO TPSVCDEF-REC.
 SET TPGETANY TO TRUE.
 CALL "TPGETRPLY" USING
        TPSVCDEF-REC TPTYPE-REC WK-AREA TPSTATUS-REC.

 IF SUB-TYPE = "cbuf"
        MOVE OUTPUT IN CREDIT-REP TO OUTPUT IN CREDIT-REQ
 ELSE
        MOVE OUTPUT IN DEBIT-REP TO OUTPUT IN DEBIT-REQ.
```

```
*
 IF REQ-SUCCEED IN DEBIT-REQ AND REQ-SUCCEED IN CREDIT-REQ
        CALL "TXCOMMIT"
 ELSE
        CALL "TXROLLBACK".
```

Service AP

```
*
* WK-AREA is where service requests are read into.
*
 01  WK-AREA PIC X(100).
*
 01  DEBIT-REC REDEFINES WK-AREA.
 COPY DEBIT.
*
 01  CREDIT-REC REDEFINES WK-AREA.
 COPY CREDIT.
*
 MOVE LENGTH OF WK-AREA TO LEN.
*
 CALL "TPSVCSTART" USING
        TPSVCDEF-REC TPTYPE-REC WK-AREA TPSTATUS-REC.
*
 IF SERVICE-NAME = "DEBIT"
        CALL "PROCESS-DEBIT" USING DEBIT-REC
 ELSE
        CALL "PROCESS-CREDIT" USING CREDIT-REC.
*
 IF UPDATE-SUCCEEDED
* DBMS update successful
        SET TPSUCCESS TO TRUE
        SET REQ-SUCCEED TO TRUE
 ELSE
* Ensure transaction rolls back
        SET TPFAIL TO TRUE
        SET REQ-FAIL TO TRUE.
*
COPY TPRETURN REPLACING DATA-REC BY WK-AREA.
```

B.2 Example 2

The following example shows the sequence of calls made by a Client Application and a Server Application that illustrates the use of the Conversational Service Paradigm in the COBOL programming language.

Client AP

```
*
* INQUIRY-REC can hold 10 inquiry records.
*
 01 INQUIRY-REC.
 COPY INQUIRY.
*
 CALL "TXBEGIN".
*
*  Issue TPCONNECT to INQUIRY.
*
 MOVE "X_COMMON" TO REC-TYPE.
 MOVE "inq_buf" TO SUB-TYPE.
 MOVE LENGTH OF INQ-REC TO LEN.
 MOVE LOW-VALUES TO TPSVCDEF-REC.
 SET TPRECVONLY TO TRUE.

 MOVE "INQUIRY" TO SERVICE-NAME.
*
 CALL "TPCONNECT" USING
        TPSVCDEF-REC TPTYPE-REC INQUIRY-REC TPSTATUS-REC.
RECV.
*
* Issue a TPRECV and process 10 records at a time.
*
 MOVE LOW-VALUES TO TPSVCDEF-REC.
 SET TPNOCHANGE TO TRUE.
 CALL "TPRECV" USING
        TPSVCDEF-REC TPTYPE-REC INQUIRY-REC TPSTATUS-REC.
*
* Format and display in AP-specific manner the account returned.
*
 IF NOT TPEEVENT
        GO TO RECV.
*
 IF TPEV-SVCSUCC
        CALL "TXCOMMIT"
 ELSE
        CALL "TXROLLBACK".
```

Service AP

```
*
 01  INQUIRY-REC.
 COPY INQUIRY.
*
* Gather input parameters about service request.
*
 CALL "TPSVCSTART" USING
        TPSVCDEF-REC TPTYPE-REC INQUIRY-REC TPSTATUS-REC.
*
* Gather from DBMS next 10 records and send them to client.
*
SEND.

 MOVE LOW-VALUES TO TPSVCDEF-REC.
 SET TPSENDONLY TO TRUE.
 CALL "TPSEND" USING
        TPSVCDEF-REC TPTYPE-REC INQUIRY-REC TPSTATUS-REC.

 IF MORE RECORDS
        GO TO SEND.

 IF INQUIRY-SUCCEEDED
        SET TPSUCCESS TO TRUE
 ELSE
        SET TPFAIL TO TRUE
*
* No data to send back with TPRETURN.
*
 MOVE SPACES TO REC-TYPE.
 COPY TPRETURN REPLACING DATA-REC BY INQUIRY-REC.
```

Appendix C

TX Extensions for the XATMI Interface

XATMI requires no extensions to the TX (Transaction Demarcation) interface.

X/Open CAE Specification

Part 4:

XATMI Application Service Element Appendix

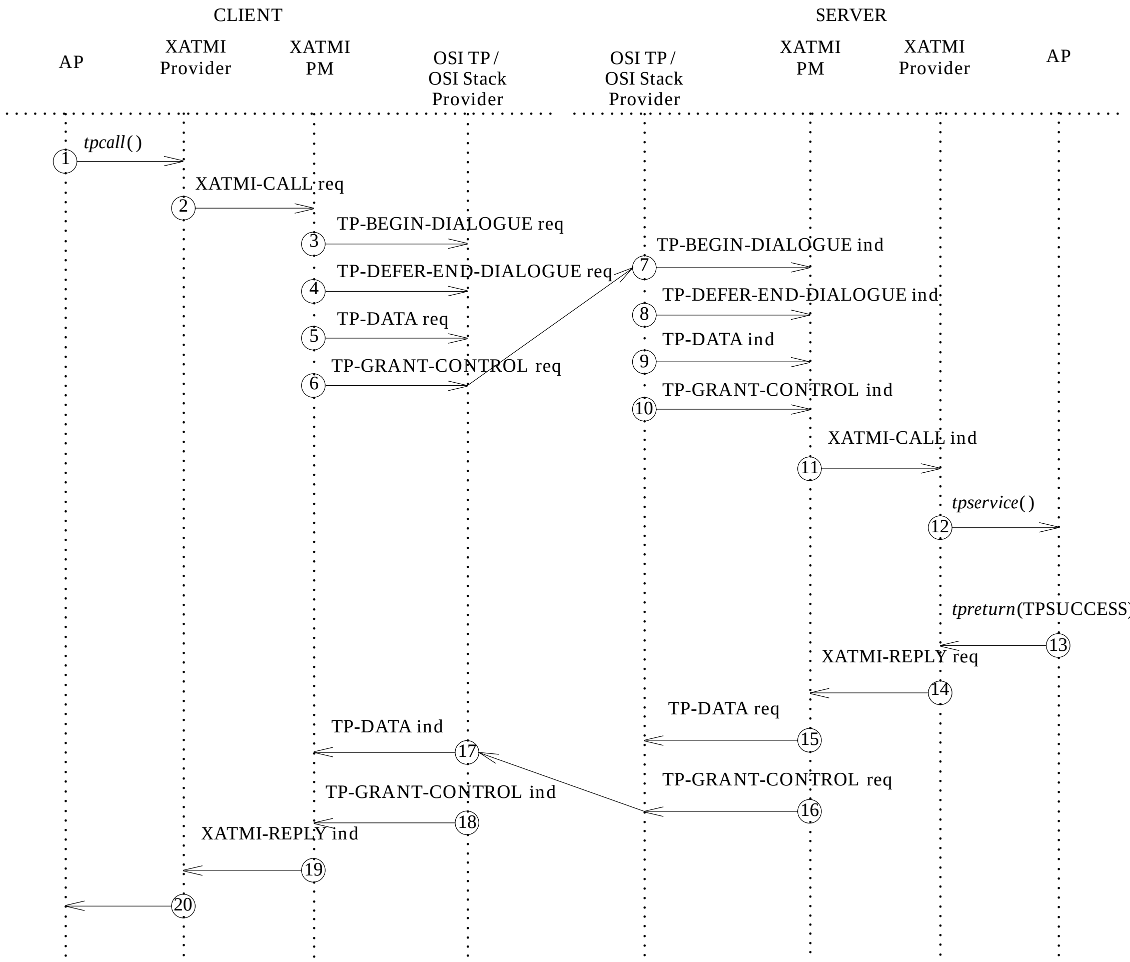
X/Open Company Ltd.

Scenarios

This appendix contains examples of the usage of XATMI-ASE. Implementors should note that although the examples may appear to be strikingly similar to existing OSI TP implementations, such as those based on the XAP-TP, these examples should not be construed as exact usage of such implementations.

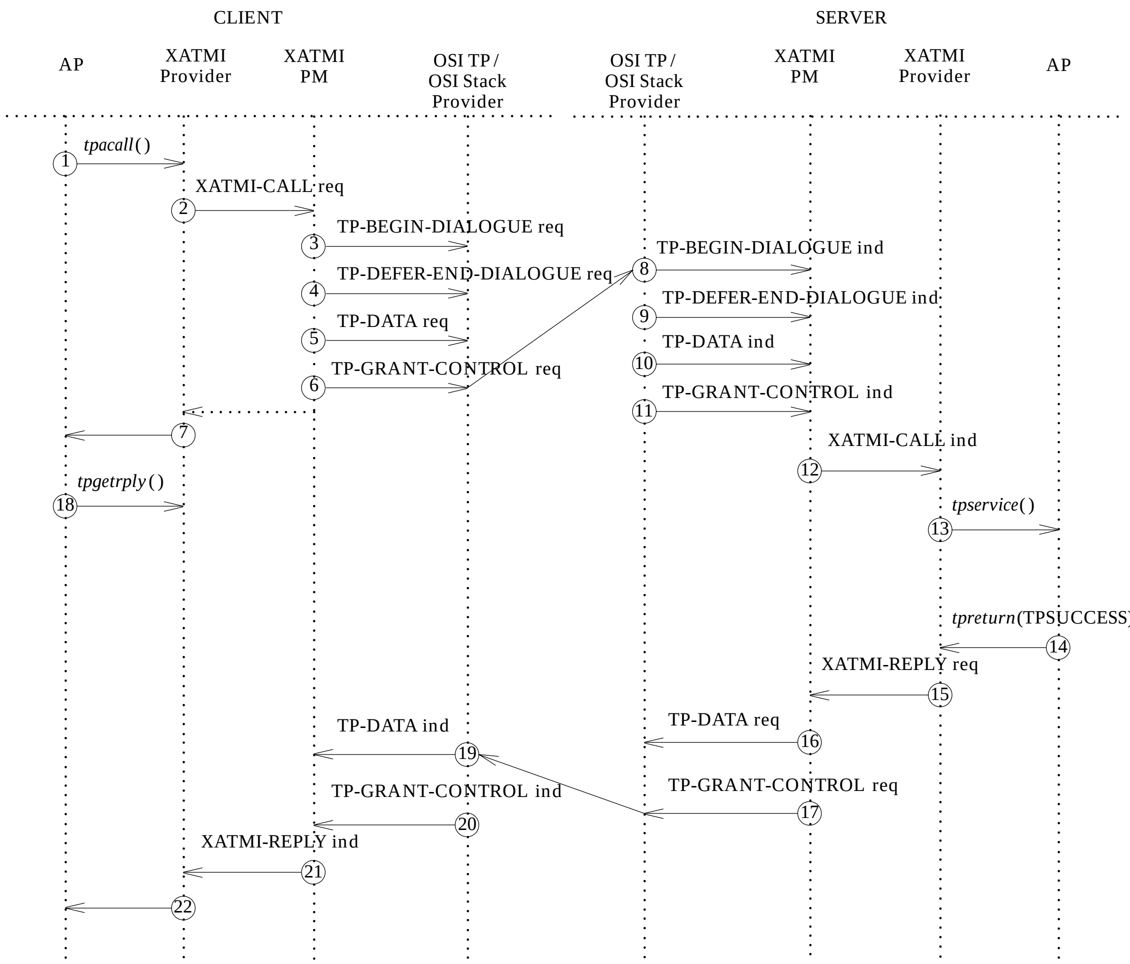
D.1 Synchronous Service Request within a Global Transaction

This scenario uses client mappings 1 and 3, and server mappings 25 and 31.



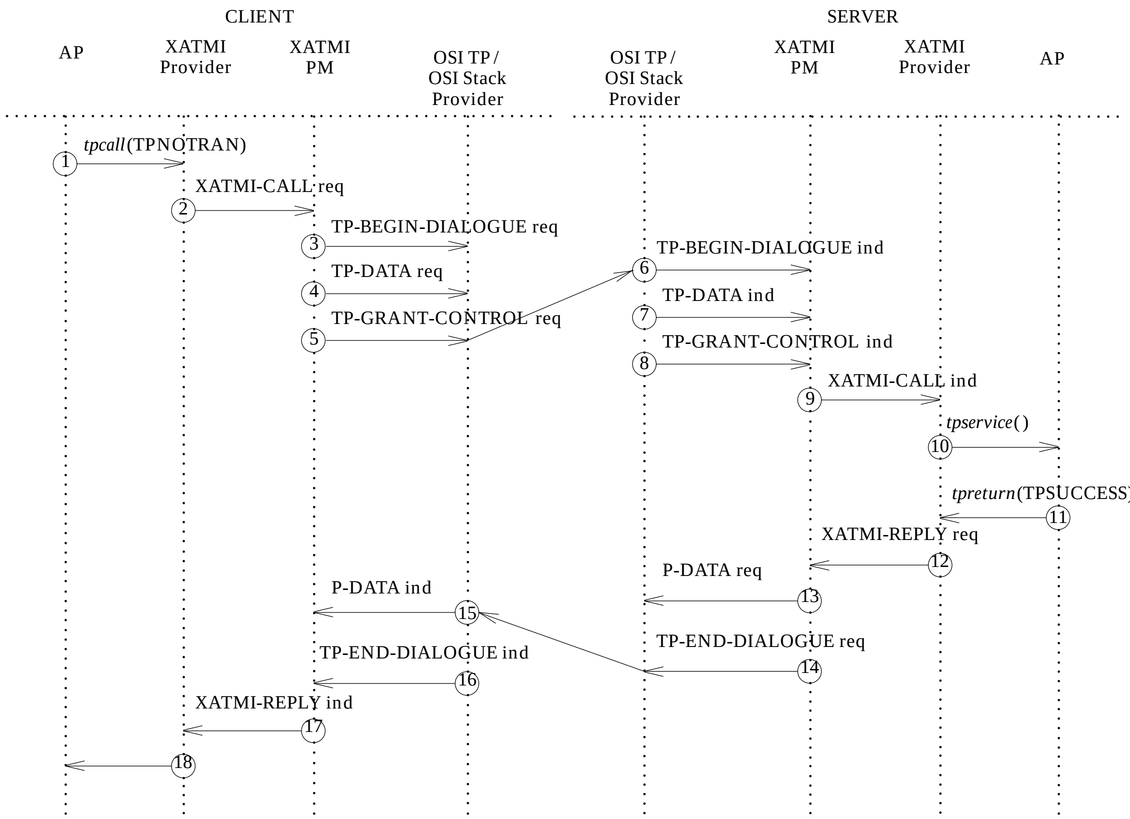
D.2 Asynchronous Service Request within a Global Transaction

This scenario uses client mappings 1 and 3, and server mappings 25 and 31.



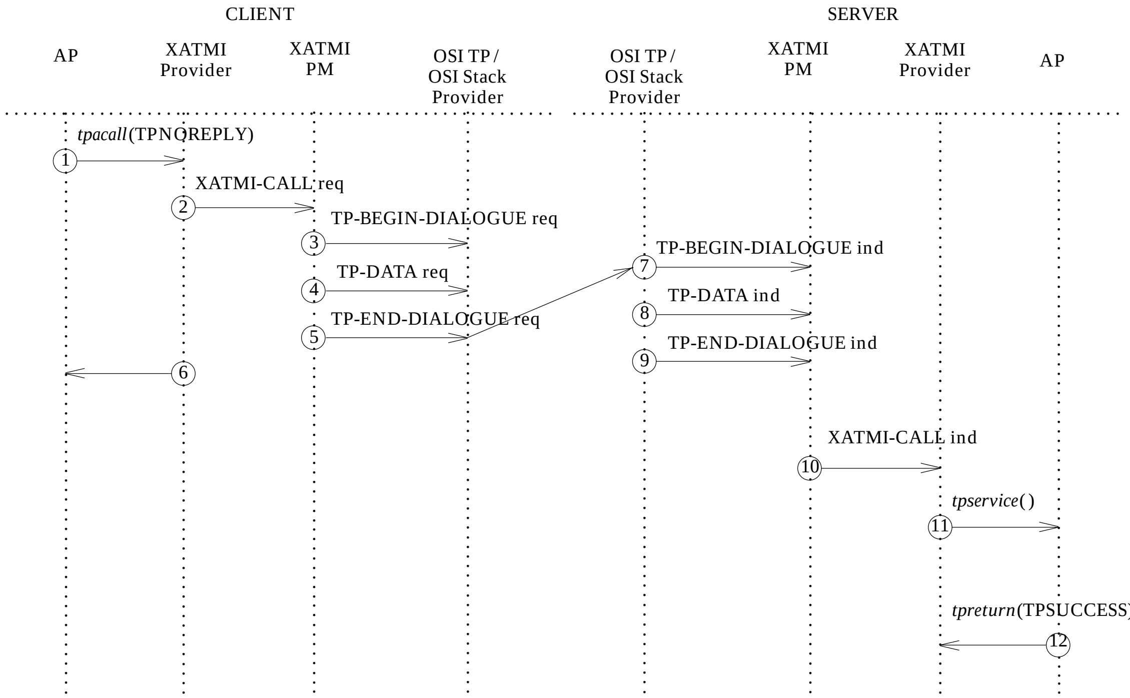
D.3 Synchronous Service Request outside any Global Transaction

This scenario uses client mappings 1 and 4, and server mappings 25 and 32.



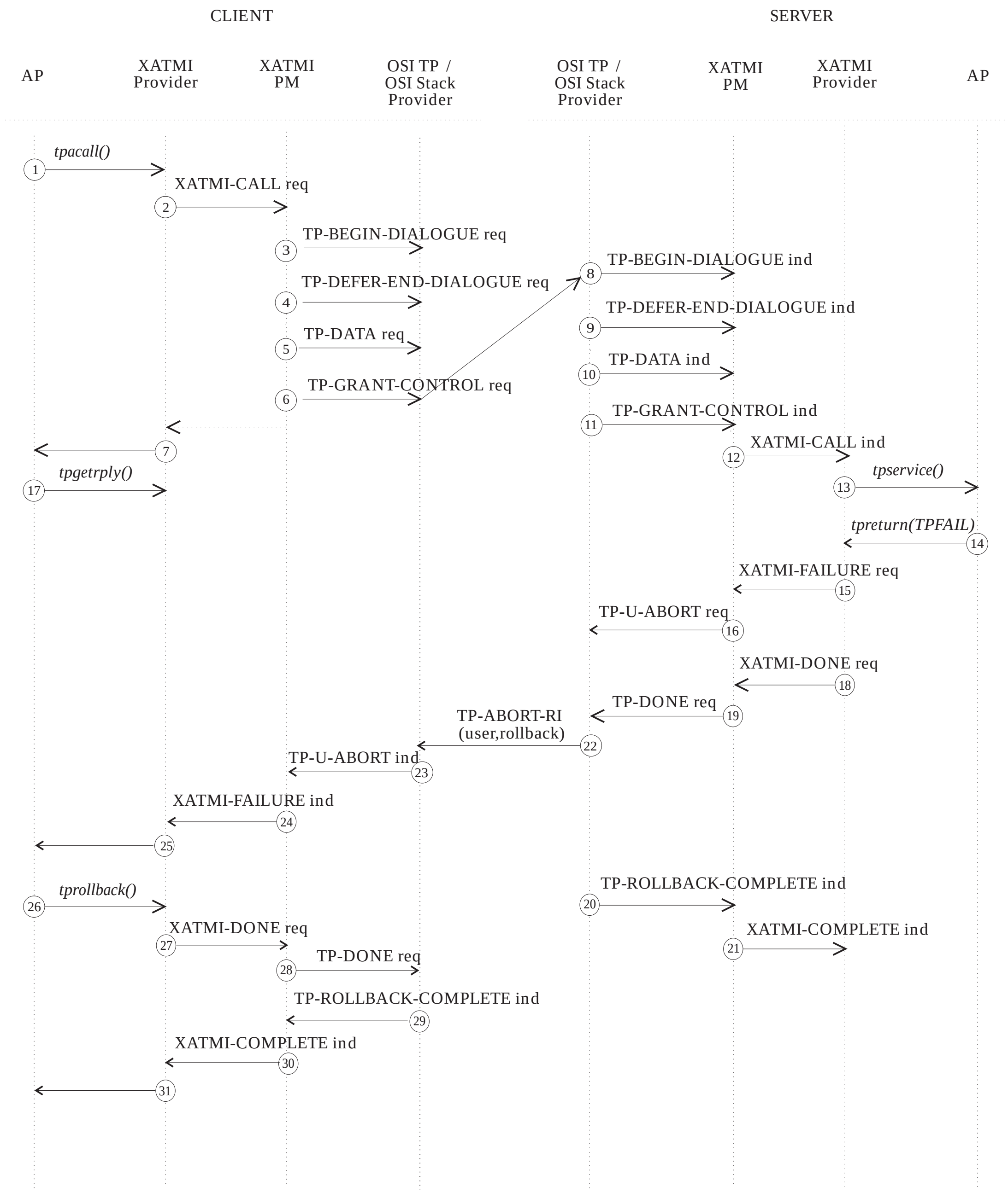
D.4 Asynchronous Service Request with No Reply

This scenario uses client mapping 2 and server mapping 27.



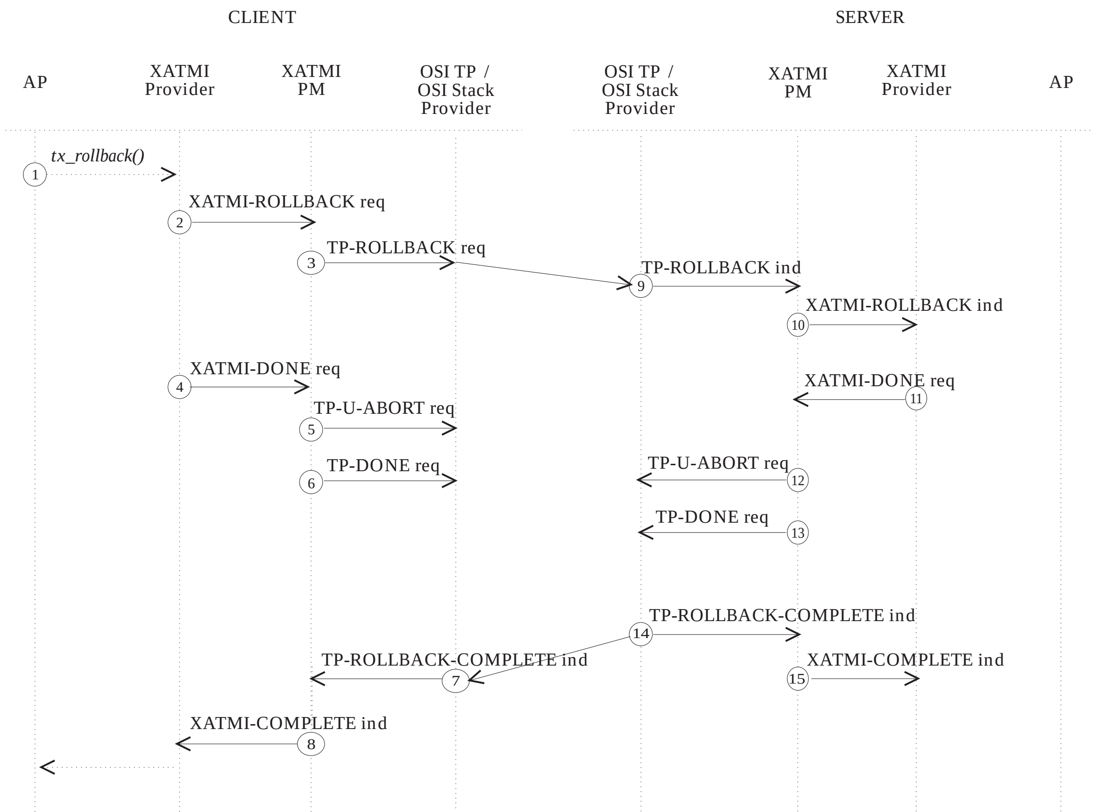
D.5 Service Return Failure within a Global Transaction

This scenario uses client mappings 1, 5, 20, 8 and 23, and server mappings 25, 33, 45, 43 and 48.



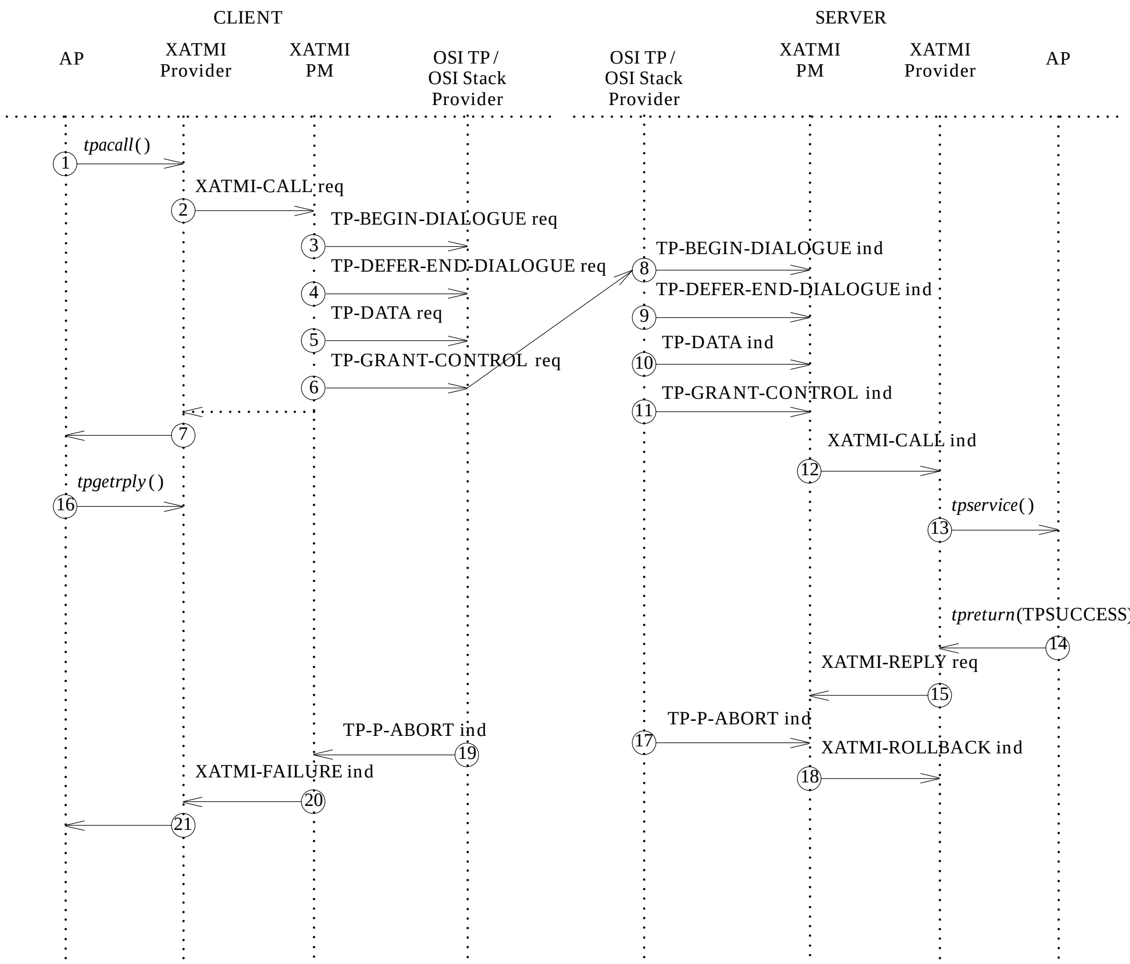
D.6 Transaction Rollback

This scenario uses client mappings 20, 19 and 23, and server mappings 42, 45 and 50.



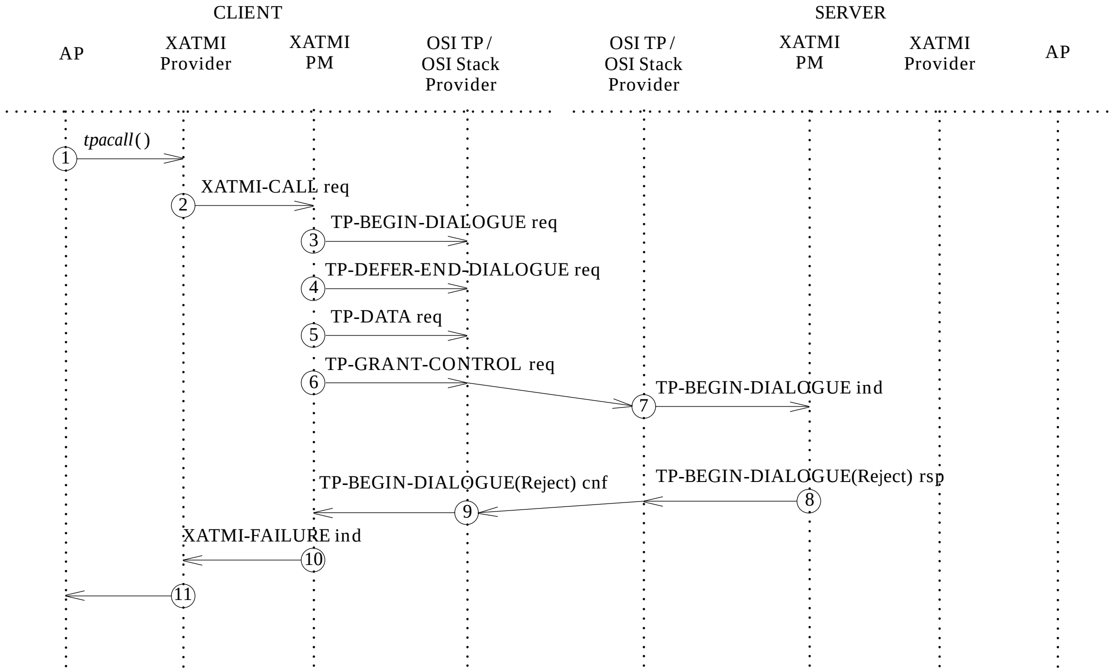
D.7 Network Failure within a Transaction

This scenario uses client mappings 1 and 6, and server mappings 25, 31 and 46.



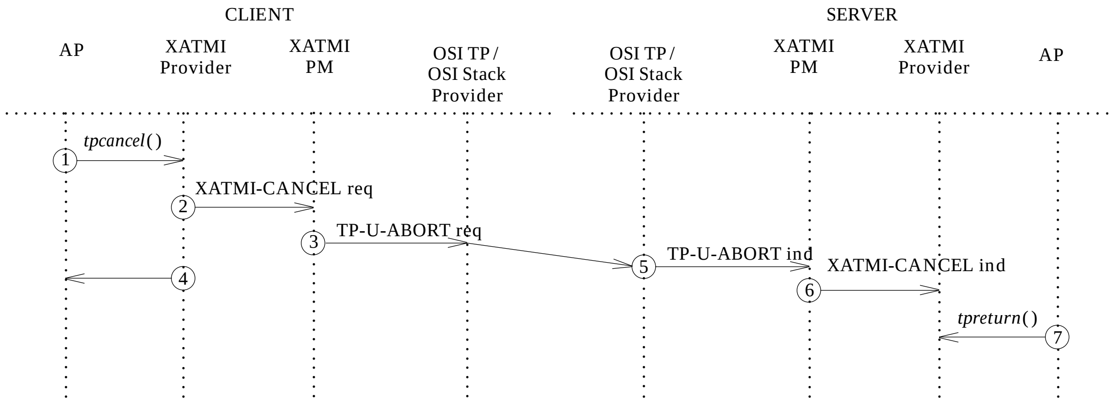
D.8 Dialogue Setup Failure

This scenario uses client mappings 1 and 7, and server mapping 27.



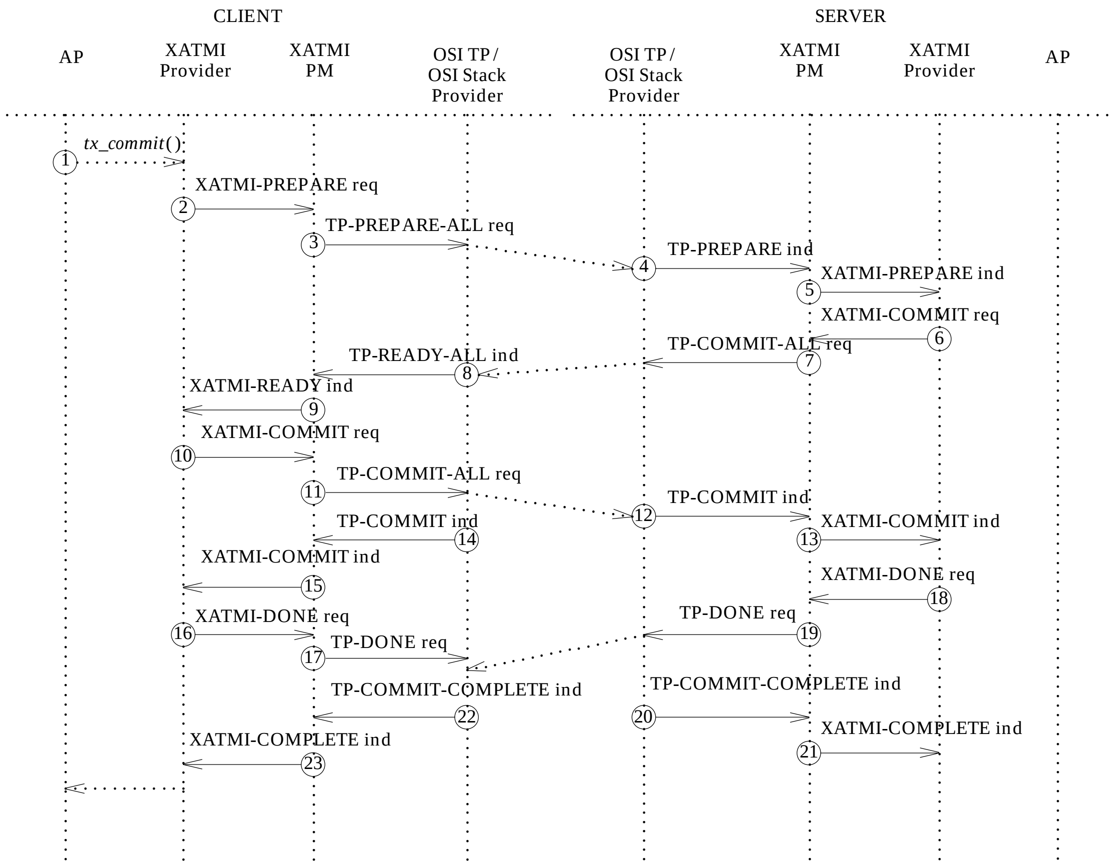
D.9 Service Request Cancel

This scenario uses client mapping 8 and server mapping 34.



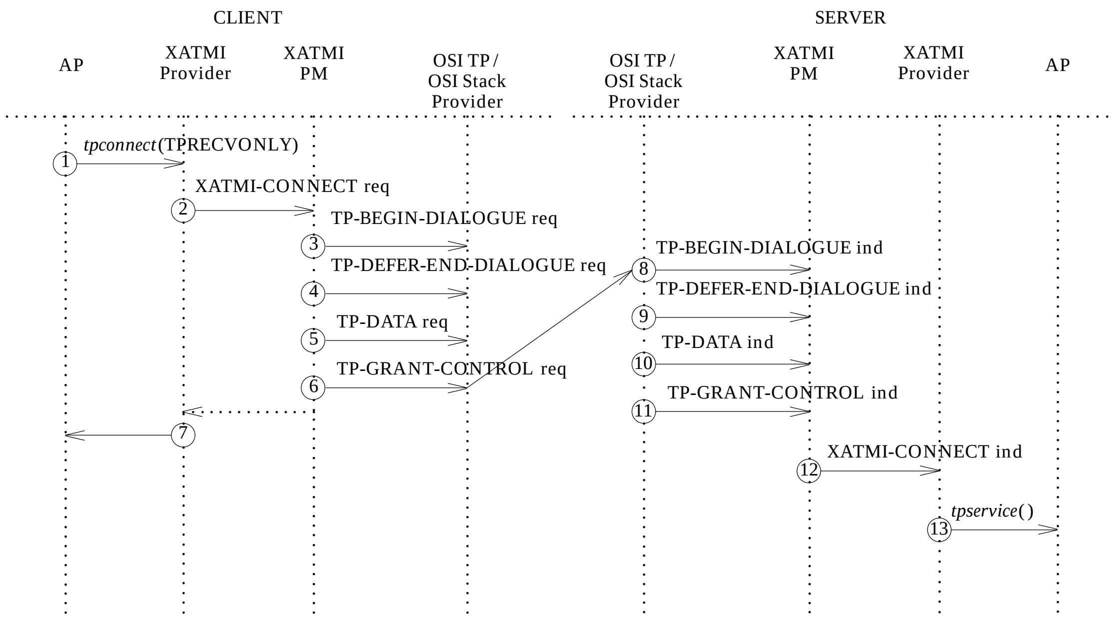
D.10 Transaction Commit

This scenario uses client mappings 14, 15, 16, 17, 18 and 22, and server mappings 40, 41, 42, 43 and 47.



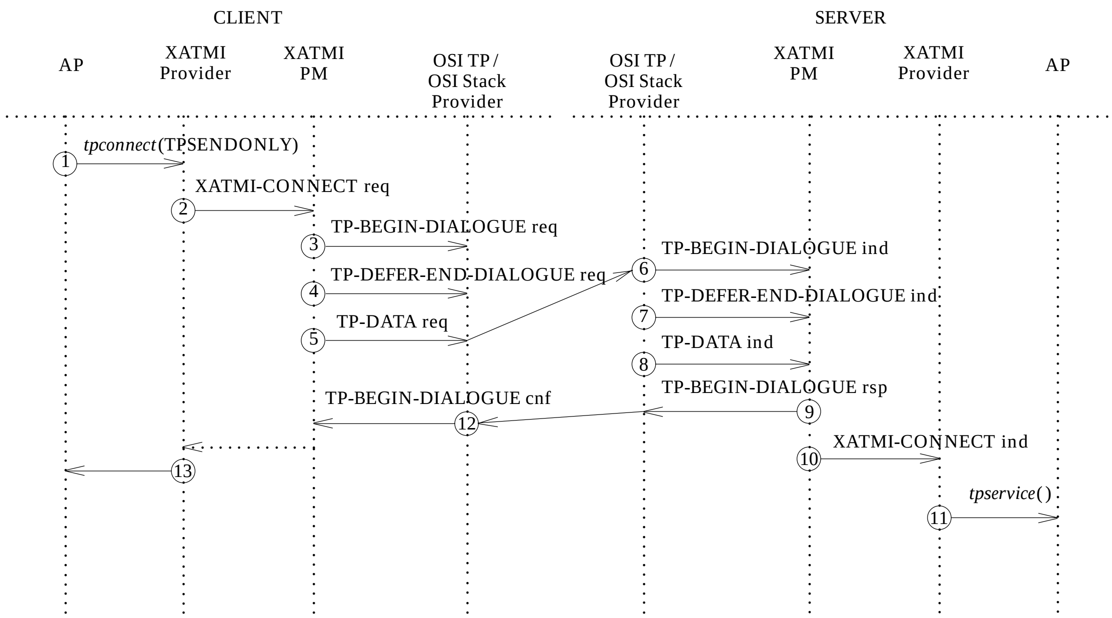
D.11 Conversational Service Request (Service Gets Control)

This scenario uses client mapping 9 and server mapping 28.



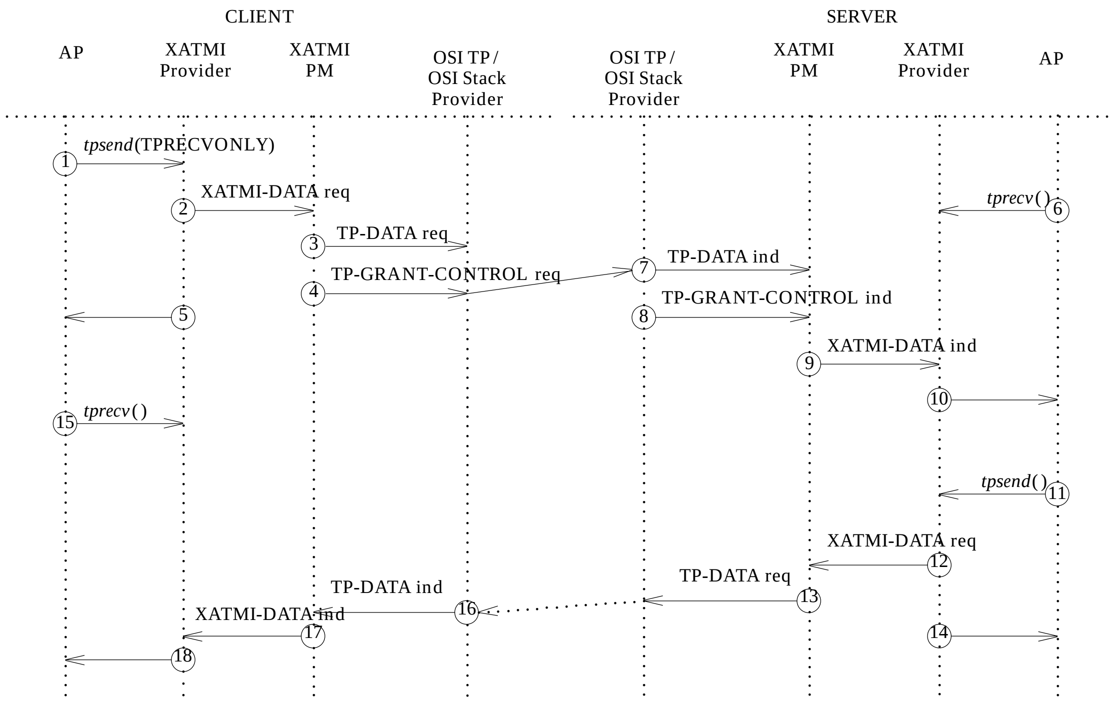
D.12 Conversational Service Request (Requester Keeps Control)

This scenario uses client mapping 10 and server mapping 29.



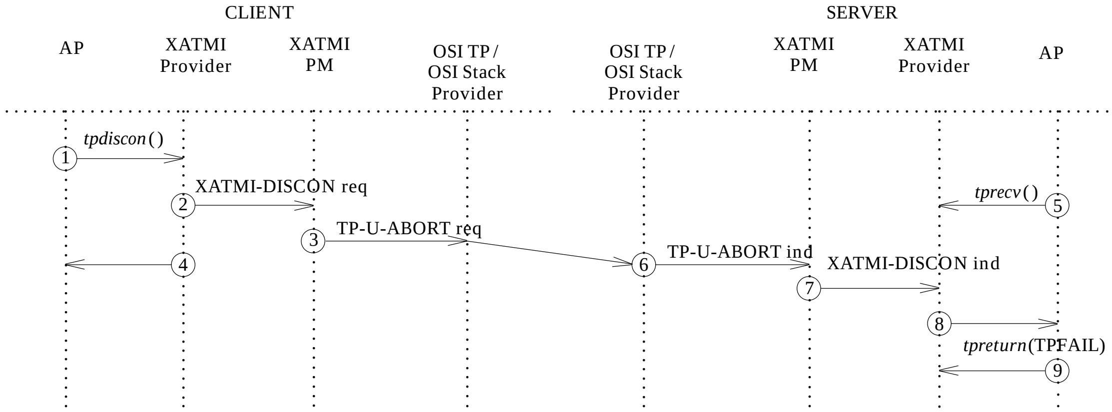
D.13 Conversational Send and Receive with Grant Control

This scenario uses client mappings 12 and 13, and server mappings 38 and 39.



D.14 Disconnection of Conversational Service

This scenario uses client mapping 11 and server mapping 36.



Index