INTERNATIONAL STANDARD

ISO/IEC 14834

First edition
1996-08-15

Information technology — Distributed Transaction Processing — The XA Specification

Technologies de l'information — Traitement transactionnel réparti — La spécification XA

Reference number
ISO/IEC 14834:1996(E)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Contents

Chapter	**1**	**General**	**1**
	1.1	Scope	1
	1.2	X/Open DTP Model	1
	1.3	Document Structure	2
	1.4	Normative References	2
Chapter	**2**	**Model and Definitions**	**3**
	2.1	X/Open DTP Model	4
	2.2	Definitions	5
	2.2.1	Transaction	5
	2.2.2	Distributed Transaction Processing	5
	2.2.3	Application Program	5
	2.2.4	Resource Manager	5
	2.2.5	Global Transactions	6
	2.2.6	Transaction Branches	6
	2.2.7	Transaction Manager	6
	2.2.8	Thread of Control	7
	2.2.9	Tightly- and Loosely-coupled Threads	7
	2.3	Transaction Completion and Recovery	8
	2.3.1	Rolling Back the Global Transaction	8
	2.3.2	Protocol Optimisations	8
	2.3.3	Heuristic Branch Completion	9
	2.3.4	Failures and Recovery	9
Chapter	**3**	**Interface Overview**	**11**
	3.1	Index to Services in the XA Interface	12
	3.2	Opening and Closing Resource Managers	13
	3.3	Association of Threads with Transaction Branches	14
	3.3.1	Registration of Resource Managers	15
	3.4	Branch Completion	16
	3.5	Synchronous, Non-blocking, and Asynchronous Modes	17
	3.6	Failure Recovery	17
Chapter	**4**	**The "xa.h"Header**	**19**
	4.1	Naming Conventions	19
	4.2	Transaction Identification	19
	4.3	Resource Manager Switch	21
	4.4	Flag Definitions	22

© ISO/IEC 1996

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

	4.5	Return Codes	24
Chapter	**5**	**Reference Manual Pages**	**27**
		ax_reg( )	28
		ax_unreg( )	31
		xa_close( )	32
		xa_commit( )	34
		xa_complete( )	37
		xa_end( )	38
		xa_forget( )	41
		xa_open( )	43
		xa_prepare( )	45
		xa_recover( )	48
		xa_rollback( )	50
		xa_start( )	53
Chapter	**6**	**State Tables**	**57**
	6.1	Resource Manager Initialisation	58
	6.2	Association of Threads of Control with Transactions	59
	6.2.1	Dynamic Registration of Threads	59
	6.3	Transaction States	61
	6.4	Asynchronous Operations	63
Chapter	**7**	**Implementation Requirements**	**65**
	7.1	Application Program Requirements	65
	7.2	Resource Manager Requirements	66
	7.2.1	The Application Program (Native) Interface	68
	7.3	Transaction Manager Requirements	69
Appendix	**A**	**Complete Text of "xa.h"**	**71**
Appendix	**B**	**DTP Model — Introduction**	**75**
	B.1	Overview	75
	B.2	Benefits of X/Open DTP	76
	B.3	Areas Not Addressed	76
	B.4	Relationship to International Standards	76
Appendix	**C**	**DTP Model — Definitions**	**77**
	C.1	Transaction Definitions	77
	C.2	Model Definitions	79
Appendix	**D**	**DTP Model — The Model**	**81**
	D.1	Functional Model	82
	D.2	Functional Components	83
	D.2.1	Application Program (AP)	83
	D.2.2	Transaction Manager (TM)	83
	D.2.3	Resource Manager (RM)	83
	D.2.4	Communication Resource Manager (CRM)	84

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.3 Interfaces between Functional Components 85
D.3.1 Functional Component Interfaces 85
D.3.2 Data Interfaces 87
D.4 Activity Involving a Single AP 88
D.4.1 Transaction Initiation 88
D.4.2 Transaction Association 88
D.4.3 Transaction Commitment 88
D.4.4 Transaction Rollback 89
D.4.5 Heuristic Transaction Completion 89
D.4.6 Recovery after Failure 90
D.5 Distributed Communication Facilities 91
D.5.1 Communication within TM Domains 91
D.5.2 Communication across TM Domains 91
D.5.3 Sharing Resources across TM Domains 91
D.5.4 Global Transaction Demarcation 91
D.5.5 Global Transaction Tree Structure 91
D.5.6 Global Transactions and the Transaction Tree 92
D.5.7 Tightly- and Loosely-coupled Threads 93
D.5.8 Commitment Coordination 93
D.6 Activity Involving Two or More APs 94
D.6.1 Transaction Initiation 94
D.6.2 Transaction Association 94
D.6.3 Transaction Commitment 94
D.6.4 Transaction Rollback 95
D.6.5 Heuristic Transaction Completion 95
D.6.6 Recovery after Failure 95
D.7 CRM Communication Paradigms with APs 96
D.7.1 The TxRPC Interface 96
D.7.2 The XATMI Interface 96
D.7.3 The CPI-C Version 2 Interface 97
D.7.4 Relationships between the Communication Paradigms 97
D.8 High-level TP Language 98

Appendix E DTP Model — Frequently Asked Questions 99

Appendix F Bibliography 103

Index 105

List of Figures

2-1 Functional Components and Interfaces 4
3-1 The XA Interface 11
C-1 A TM Domain with Four Instances 79
D-1 Functional Components and Interfaces 82
D-2 Global Transaction Tree Structure 92
E-1 Projection of Model onto Processes 101

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

List of Tables

4-1	Flags used in Particular Function Calls	23
6-1	State Table for Resource Manager Initialisation	58
6-2	State Table for Transaction Branch Association	59
6-3	State Table for Transaction Branch Association (Dynamic Registration)	60
6-4	State Table for Transaction Branches	62
6-5	State Table for Asynchronous Operations	63

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 14834 was prepared by X/Open Company Ltd. (as XO/CAE/91/300) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Appendix A forms an integral part of this International Standard. Appendices B to F are for information only.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

(This introduction is not a normative part of ISO/IEC 14834, Information technology—Distributed Transaction Processing—The XA Specification, but is included for information only.)

This International Standard specifies the bidirectional interface between a transaction manager and resource manager (the XA interface) in an X/Open Distributed Transaction Processing (DTP) environment. It is based on X/Open CAE Specification, Distributed Transaction Processing: The XA Specification (December 1991). This International Standard is technically identical to the X/Open version. For informative purposes, this International Standard also contains the text of the X/Open DTP Reference Model Version 3 which X/Open has published as a separate Guide.

Typographical Conventions

The following typographical conventions are used throughout this document:

- `Constant width` strings are code examples or literals and are to be typed just as they appear.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics also denote:
  - — variable names
  - — commands or utilities
  - — functions; these are shown as follows: *name*( ).
- The notation **"file.h"** indicates a header.
- The notation [ABCD] is the name of a return value.
- Ellipses (. . .) are used to show that additional arguments are optional.

Trademarks

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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Information technology — Distributed Transaction Processing — The XA Specification

Chapter 1: General

1.1 Scope

This International Standard specifies the *XA interface*: the bidirectional interface between a transaction manager and a resource manager in an X/Open Distributed Transaction Processing (DTP) environment. The XA interface is not an ordinary Application Programming Interface (API); it is a system-level interface between DTP software components.

This International Standard is technically identical to X/Open CAE Specification, Distributed Transaction Processing: The XA Specification (December 1991). Like that specification, this International Standard does not define the full aspects of the DTP model that pertain to communication.

1.2 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 full 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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- 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.

1.3 Document Structure

Relevant definitions and other important concepts that pertain to this International Standard are discussed in Chapter 2. That chapter also defines the AP, TM, and RM in more detail, and describes their interaction. Chapter 3 is an overview of the XA interface, describing the situations in which each of the services is used. Chapter 4 discusses the data structures that are part of the XA interface. Reference manual pages for each routine in the XA interface are presented in Chapter 5; state tables follow in Chapter 6. Chapter 7 summarises the implications of this International Standard on the implementors of RMs and TMs; it also identifies features that are optional. Appendix A presents the contents of an "**xa.h**" header file in both ANSI C and Common Usage C. Appendix F contains a bibliography.

For informative purposes, this International Standard also contains the text of the X/Open DTP Reference Model Version 2 (November 1993) which X/Open publishes as a separate Guide. (See Appendix B, Appendix C, Appendix D, and Appendix E.)

1.4 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

1. ISO/IEC 8824:1990, *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1).*
2. ISO/IEC 8825:1990, *Information technology—Open Systems Interconnection—Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).*
3. ISO/IEC 9804:1994, *Information technology—Open Systems Interconnection—Service definition for the commitment, concurrency and recovery service element.*
4. ISO/IEC 9805-1:1994, *Information technology—Open Systems Interconnection—Protocol for the Commitment, Concurrency and Recovery service element: Protocol Specification.*
5. ISO/IEC 9899:1990, *Programming languages—C.*
6. ISO/IEC 10026-1:1992, *Information technology—Open Systems Interconnection—Distributed Transaction Processing—Part 1: OSI TP Model.*
7. ISO/IEC 10026-2:1996, *Information technology—Open Systems Interconnection—Distributed Transaction Processing—Part 2: OSI TP Service.*
8. ISO/IEC 10026-3:1996, *Information technology—Open Systems Interconnection—Distributed Transaction Processing—Part 3: Protocol Specification.*

See Appendix F for bibliographic references.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Chapter 2: Model and Definitions

This chapter discusses the XA interface in general terms and provides necessary background material for the rest of this International Standard. 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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.1 X/Open DTP Model

The boxes in Figure 2-1 are the functional components and the connecting lines are the interfaces between them. The arrows indicate the directions in which control may flow.

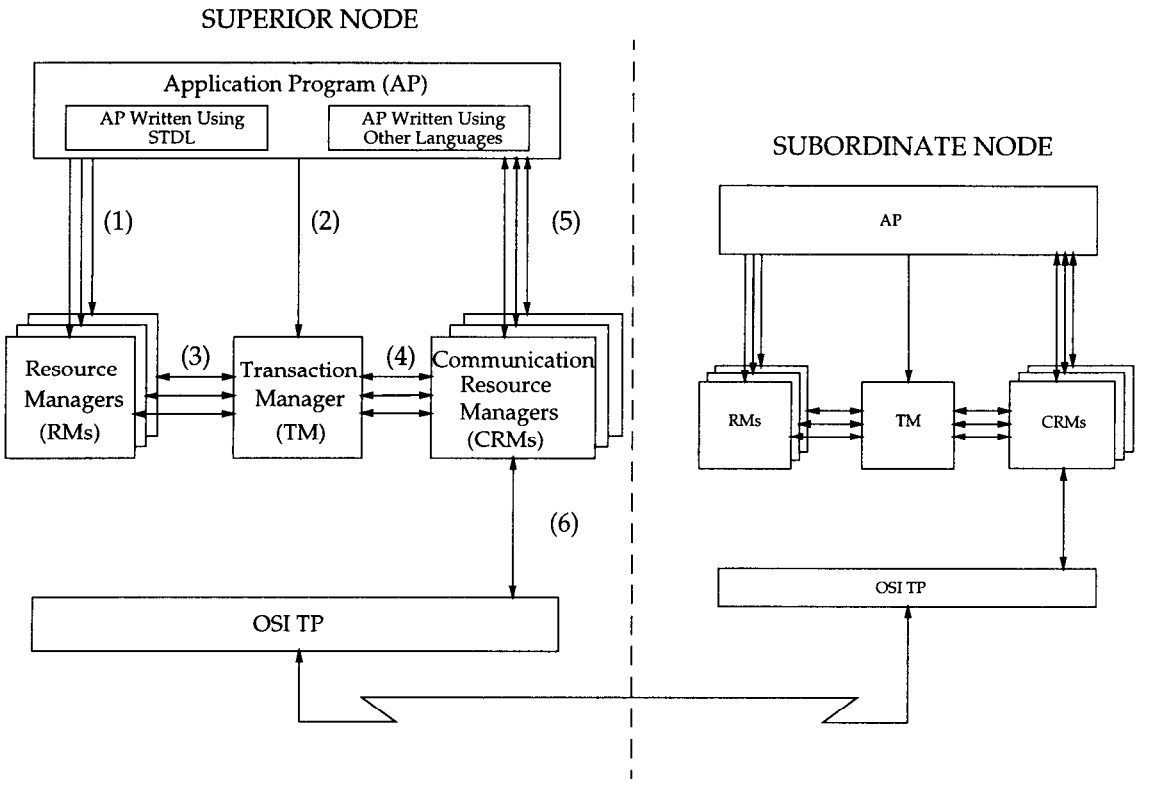


Figure 2-1 Functional Components and Interfaces

The numbers in brackets in Figure 2-1 represent the different X/Open interfaces that are used in the DTP model. The subject of this International Standard is interface (3): the XA interface by which TMs and RMs interact. Descriptions of the functional components relevant to this International Standard can be found in Section 2.2 on page 5. For more details of the the DTP model as shown in Figure 2-1, including definitions of all components and interfaces, see Appendix B, Appendix C, Appendix D, and Appendix E.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.2 Definitions

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 communications. A typical transaction modifies shared resources. (The referenced OSI TP standard (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 that is rolled back at any time.

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 Distributed Transaction Processing

Within the scope of this International Standard, 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 a standard interface, such as Structured Query Language (SQL), it must also address these two items to be useful in the DTP environment.

2.2.3 Application Program

The AP defines transactions and accesses resources within transaction boundaries. Each AP specifies a sequence of operations that involves resources such as terminals and databases. This International Standard generally uses the term AP to refer to a single instance of an application program.

2.2.4 Resource Manager

An RM manages a certain part of the computer's shared resources. Many other software entities can request access to the resource from time to time, using services that the RM provides. Here are some examples of RMs:

- A database management system (DBMS) is an RM. Typical DBMSs are capable of defining transactions and committing work atomically.
- A file access method such as the Indexed Sequential Access Method (ISAM) can be the basis for an RM. Typically, an ISAM RM must be enhanced to support transactions as defined herein.
- A print server might be implemented as an RM.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A single RM may service multiple independent resource domains. An RM *instance* services one of these domains. (See also Section 3.2 on page 13.) Unless specified otherwise, operations this International Standard allows on an RM are allowed on each RM instance.

2.2.5 Global Transactions

Every RM in the DTP environment must support transactions as described in Section 2.2.1 on page 5. 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* (which is normally equivalent to an OSI TP transaction when OSI TP is used as the communication protocol; see the referenced OSI TP standards). 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.

2.2.6 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 (see Section 2.3 on page 8). Each of the RM's internal units of work in support of a global transaction is part of exactly one branch. (The term *transaction branch* does not necessarily have the same meaning as in OSI TP. The concept is the same, but the parties may be different; see the referenced OSI TP standards.)

A global transaction might have more than one branch when, for example, the AP uses multiple processes or is involved in the same global transaction by multiple remote APs.

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.

Each transaction branch identifier (or XID — see Section 4.2 on page 19) that the TM gives the RM identifies both a global transaction and a specific branch. The RM may use this information to optimise its use of shared resources and locks.

2.2.7 Transaction Manager

TMs manage global transactions, coordinate the decision to commit them or roll them back, and coordinate failure recovery. The AP defines the start and end of a global transaction by calling a TM. The TM assigns an identifier to the global transaction (see Section 4.2 on page 19). The TM manages global transactions and informs each RM of the XID on behalf of which the RM is doing work. Although RMs can manage their own recoverable work units as they see fit, each RM must accept XIDs and associate them with those work units. In this way, an RM knows what recoverable work units to complete when the TM completes a global transaction.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.2.8 Thread of Control

A thread of control (or a *thread*) is the entity, with all its context, that is currently in control of a processor. A thread of control is an operating-system process: an address space and single thread of control that executes within that address space, and its required system resources. The context may include the process' locks on shared resources, and the files the process has open. For portability reasons, the notion of thread of control must be common among the AP, TM, and RM.

The thread concept is central to the TM's coordination of RMs. APs call RMs to request work, while TMs call RMs to delineate transaction branches. The way the RM knows that a given work request pertains to a given branch is that the AP and the TM both call it from *the same thread of control*. For example, an AP thread calls the TM to declare the start of a global transaction. The TM records this fact and informs RMs. After the AP regains control, it uses the native interface of one or more RMs to do work. The RM receives the calls from the AP and TM in the same thread of control.

Certain XA routines, therefore, must be called from a particular thread. The reference manual pages in Chapter 5 indicate which routines require this.

2.2.9 Tightly- and Loosely-coupled Threads

Many application threads of control can participate in a single global transaction. All the work done in these threads is atomically completed. Within a single global transaction, the relationship between any pair of participating threads is either *tightly-coupled* or *loosely-coupled*:

- A tightly-coupled relationship is one where a pair of threads are designed to share resources. In addition, with respect to an RM's isolation policies, the pair are treated as a single entity. Thus, for a pair of tightly-coupled threads, the RM must guarantee that resource deadlock does not occur within the transaction branch.
- A loosely-coupled relationship provides no such guarantee. With respect to an RM's isolation policies, the pair may be treated as if they were in separate global transactions even though the work is atomically completed.

Within a single global transaction, a set of tightly-coupled threads may consist of more than just a pair. Moreover, many sets of tightly-coupled threads may exist within the same global transaction and each set is loosely coupled with respect to the others. The reference manual pages in Chapter 5 indicate how a TM communicates these relationships to an RM.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.3 Transaction Completion and Recovery

TMs and RMs use two-phase commit with presumed rollback, as defined by the referenced OSI TP standard (model).

In Phase 1, the TM asks all RMs to *prepare to commit* (or *prepare*) transaction branches. This asks whether the RM can guarantee its ability to commit the transaction branch. An RM may have to query other entities internal to that RM.

If an RM can commit its work, it records stably the information it needs to do so, then replies affirmatively. A negative reply reports failure for any reason. After making a negative reply and rolling back its work, the RM can discard any knowledge it has of the transaction branch.

In Phase 2, the TM issues all RMs an actual request to commit or roll back the transaction branch, as the case may be. (Before issuing requests to commit, the TM stably records the fact that it decided to commit, as well as a list of all involved RMs.) All RMs commit or roll back changes to shared resources and then return status to the TM. The TM can then discard its knowledge of the global transaction.

2.3.1 Rolling Back the Global Transaction

The TM rolls back the global transaction if any RM responds negatively to the Phase 1 request, or if the AP directs the TM to roll back the global transaction. Therefore, any negative response vetoes the global transaction. A negative response concludes an RM's involvement in the global transaction.

The TM effects Phase 2 by telling all RMs to roll back transaction branches. They must not let any changes to shared resources become permanent. The TM does not issue Phase 2 requests to RMs that responded negatively in Phase 1. The TM does not need to record stably the decision to roll back nor the participants in a rolled back global transaction.

2.3.2 Protocol Optimisations

- **Read-only**
  An RM can respond to the TM's prepare request by asserting that the RM was not asked to update shared resources in this transaction branch. This response concludes the RM's involvement in the transaction; the Phase 2 dialogue between the TM and this RM does not occur. The TM need not stably record, in its list of participating RMs, an RM that asserts a read-only role in the global transaction.

  However, if the RM returns the read-only optimisation before all work on the global transaction is prepared, global *serialisability*[1] cannot be guaranteed. This is because the RM may release transaction context, such as read locks, before all application activity for that global transaction is finished.

1. Serialisability is a property of a set of concurrent transactions. For a serialisable set of transactions, at least one serial sequence of the transactions exists that produces identical results, with respect to shared resources, as does concurrent execution of the transaction.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- **One-phase Commit**
  A TM can use one-phase commit if it knows that there is only one RM anywhere in the DTP system that is making changes to shared resources. In this optimisation, the TM makes its Phase 2 commit request without having made a Phase 1 prepare request. Since the RM decides the outcome of the transaction branch and forgets about the transaction branch before returning to the TM, there is no need for the TM to record stably these global transactions and, in some failure cases, the TM may not know the outcome.

2.3.3 Heuristic Branch Completion

Some RMs may employ heuristic decision-making: an RM that has prepared to commit a transaction branch may decide to commit or roll back its work independently of the TM. It could then unlock shared resources. This may leave them in an inconsistent state. When the TM ultimately directs an RM to complete the branch, the RM may respond that it has already done so. The RM reports whether it committed the branch, rolled it back, or completed it with mixed results (committed some work and rolled back other work).

An RM that reports heuristic completion to the TM must not discard its knowledge of the transaction branch. The TM calls the RM once more to authorise it to forget the branch. This requirement means that the RM must notify the TM of all heuristic decisions, even those that match the decision the TM requested. The referenced OSI TP standards (model) and (service) define heuristics more precisely.

2.3.4 Failures and Recovery

A useful DTP system must be able to recover from a variety of failures. A storage device or medium, a communication path, a node, or a program could fail.

Failures that a node can correct internally might not affect a global transaction.

Failures that do not disrupt the commitment protocol let the DTP system respond by rolling back appropriate global transactions. For example, an RM recovering from a failure responds negatively to a prepare request based on the fact that it does not recognise the XID.

More significant failures may disrupt the commitment protocol. The TM typically senses the failure when an expected reply does not arrive.

Failure and recovery processing in an X/Open DTP system is compatible with the referenced OSI TP standards, which define the presumed-rollback protocol. The X/Open DTP model makes these assumptions:

- TMs and RMs have access to stable storage
- TMs coordinate and control recovery
- RMs provide for their own restart and recovery of their own state. On request, an RM must give a TM a list of XIDs that the RM has prepared for commitment or has heuristically completed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Chapter 3: Interface Overview

This chapter gives an overview of the XA interface. This is the interface between the TM and the RM in an X/Open DTP system. Chapter 5 contains reference manual pages for each routine in alphabetical order. That chapter contains C-language function prototypes.

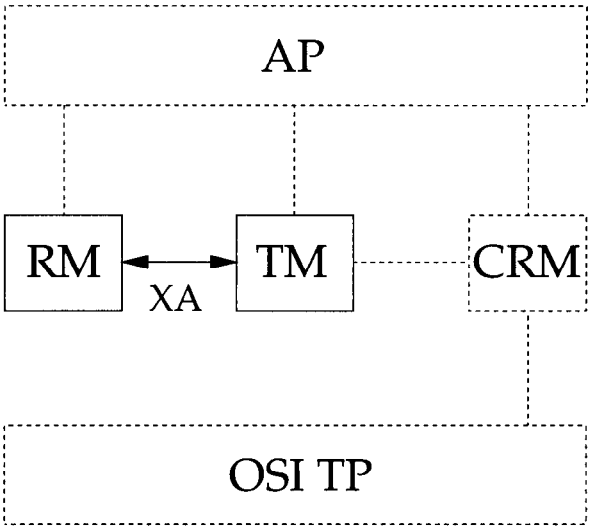


Figure 3-1 The XA Interface

The X/Open DTP model envisages interfaces between each of the AP, RM, TM, CRM, and OSI TP (see Section 2.1 on page 4). Generally, each use of the XA interface is prompted by the AP calling the TM or the RM.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1 Index to Services in the XA Interface

Name	Description	See
ax_reg	Register an RM with a TM.	Section 3.3.1 on page 15
ax_unreg	Unregister an RM with a TM.	Section 3.3.1 on page 15
xa_close	Terminate the AP's use of an RM.	Section 3.2 on page 13
xa_commit	Tell the RM to commit a transaction branch.	Section 3.4 on page 16
xa_complete	Test an asynchronous *xa_* operation for completion.	Section 3.5 on page 17
xa_end	Dissociate the thread from a transaction branch.	Section 3.3 on page 14
xa_forget	Permit the RM to discard its knowledge of a heuristically-completed transaction branch.	Section 3.4 on page 16
xa_open	Initialise an RM for use by an AP.	Section 3.2 on page 13
xa_prepare	Ask the RM to prepare to commit a transaction branch.	Section 3.4 on page 16
xa_recover	Get a list of XIDs the RM has prepared or heuristically completed.	Section 3.6 on page 17
xa_rollback	Tell the RM to roll back a transaction branch.	Section 3.4 on page 16
xa_start	Start or resume a transaction branch - associate an XID with future work that the thread requests of the RM.	Section 3.3 on page 14

The *ax_*()* routines let an RM call a TM. All TMs must provide these routines. These routines let an RM dynamically control its participation in a transaction branch.

The *xa_*()* routines are supplied by RMs operating in the DTP environment and called by TMs. When an AP calls a TM to start a global transaction, the TM shall use the XA interface to inform RMs of the transaction branch. After the AP uses the RM's native interface to do work in support of the global transaction, the TM calls *xa_*()* routines to commit or roll back branches. One other *xa_*()* routine helps the TM coordinate failure recovery.

A TM must call the *xa_*()* routines in a particular sequence (see the state tables in Chapter 6). When a TM invokes more than one RM with the same *xa_*()* routine, it can do so in an arbitrary sequence.

Note: The routine names in the *xa_*()* series are only templates.

The actual names of these functions are internal to the RM. The RM publishes the name of a structure (see Section 4.3 on page 21) that specifies the entry points to the RM. (See **Public Information** in Section 7.2 on page 67.)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.2 Opening and Closing Resource Managers

In each thread of control, the TM must call *xa_open*( ) for each RM directly accessible by that thread before calling any other *xa_**( ) routine. The TM must eventually call *xa_close*( ) to dissociate the AP from the RM.

If an RM needs to take start-up actions (such as opening files, opening paths to a server, or resynchronising a node on the network), then it could do so when called by *xa_open*( ). This International Standard does not specify the actual meaning of *xa_open*( ) and *xa_close*( ) to an RM, but the effect must be internal to the RM and must not affect transaction processing in either the calling TM or in other RMs.

If an RM requires or accepts parameters to govern its operation (for example, a directive to open files for reading only), or to identify a target resource domain, then a string argument to *xa_open*( ) conveys this information. If the RM does not require initialisation parameters, the string is typically an empty string. The *xa_close*( ) call likewise takes a string.

TMs typically read the initialisation string from a configuration file. The *xa_open*( ) routine, and the string form of its argument, support portability. A TM can give the administrator control over every run-time option that any RM provides through *xa_open*( ) with no reprogramming or relinking. The administrator must only edit a configuration file or perform a comparable, system-specific procedure.

The TM calls *xa_open*( ) with an identifier that the TM uses subsequently to identify the RM instance. A single RM may service multiple resource domains using multiple RM instances, if each instance supports independent transaction completion. For example, a single database system might access several data domains, or a single printer spooler might service multiple printers. The TM calls such an RM's *xa_open*( ) routine several times, once for each instance, using string parameters that identify the respective resource. It must generate a different RM identifier for each call.

To enhance portability, RMs in the DTP environment should rely on the use of *xa_open*( ) in place of any non-standard *open* service the RM may provide in its native interface. If an RM lets DTP applications call the native *open* routine, the effect must not conflict with the TM's use of *xa_open*( ).

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3 Association of Threads with Transaction Branches

Several threads may participate in a single transaction branch, some more than once. The *xa_start*( ) and *xa_end*( ) routines pass an XID to an RM to associate or dissociate the calling thread with a branch. The association is not necessarily the thread's initial association with the branch; its dissociation is not necessarily the final one.

A thread's association with a transaction branch can be active or suspended:

- A thread is actively associated with a transaction branch if it has called *xa_start*( ) and has not made a corresponding call to *xa_end*( ). A thread is allowed only one active association with each RM at a time.
- Certain calls to *xa_end*( ) suspend the thread's association (see **Suspend** below). The call may indicate that the association can *migrate*, that is, that any thread may resume the association. In this case, the calling thread is no longer associated. (An RM may indicate that it does not support association migration.)

If a thread calls *xa_end*( ) to suspend its association but the association cannot migrate to another thread, the calling thread retains a suspended association with the transaction branch.

Several uses of *xa_start*( ) and *xa_end*( ) are considered below:

- **Start**
  The primary use of *xa_start*( ) is to register a new transaction branch with the RM. This marks the start of the branch. Subsequently, the AP, *using the same thread of control*, uses the RM's native interface to do useful work. All requests for service made by the same thread are part of the same branch until the thread dissociates from the branch (see below).

  The return code from *xa_start*( ) may indicate that the RM has already vetoed commitment of the transaction branch. This return code is not an error; rolled back global transactions may be routine, while actual errors deserve the administrator's attention.

- **Join**
  Another use of *xa_start*( ) is to join an existing transaction branch. TMs must use a certain form of *xa_start*( ) so that RMs can validate that they recognise the passed XID.

  RMs in the DTP environment shall anticipate that many threads will try to use them concurrently. If multiple threads use an RM on behalf of the same XID, the RM is free to serialise the threads' work in any way it sees fit. For example, an RM may block a second or subsequent thread while one is active.

- **Resume**
  A special form of *xa_start*( ) associates a thread with an existing transaction branch that has been suspended (see below).

- **End**
  A typical call to *xa_end*( ) dissociates the calling thread from the transaction branch and lets the branch be completed (see Section 3.4 on page 16). Alternatively, a thread may use *xa_start*( ) to rejoin the branch.

- **Suspend**
  A form of *xa_end*( ) suspends, instead of ending, a thread's association with the transaction branch. This indicates that the thread has left the branch in an incomplete state. By using the **resume** form of *xa_start*( ), it or another thread resumes its association with the branch. Instead of resuming, the TM may completely end the suspended association by using *xa_end*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- **Rollback-only**
  An RM need not wait for global transaction completion to report an error. The RM can return **rollback-only** as the result of any *xa_start*( ) or *xa_end*( ) call. The TM can use this knowledge to avoid starting additional work on behalf of the global transaction. An RM can also unilaterally roll back and forget a transaction branch any time before it prepares it. A TM detects this when an RM subsequently indicates that it does not recognise the XID.
- **Transaction branch states**
  Several state tables appear in Chapter 6. Each call to *xa_start*( ) or *xa_end*( ) may affect the status of the thread's association with a transaction branch (see Table 6-2 on page 59 and Table 6-3 on page 60) and the status of the branch itself (see Table 6-4 on page 62). A TM must use these routines so that each thread of control makes calls in a sequence that complies with both tables.

Transaction Context

Transaction context is RM-specific information visible to the AP. The RM shall preserve certain transaction context on *xa_end*( ) so that the RM can restore context in the join or resume cases (defined above). In the join case, the RM shall make available enough transaction context so that tightly-coupled threads are not susceptible to resource deadlock within the transaction branch. In the resume case, the RM shall make available at least that RM-specific transaction context present at the time of the suspend, as if the thread had effectively never been suspended, except that other threads in the global transaction may have affected this context.

3.3.1 Registration of Resource Managers

Normally, a TM involves all associated RMs in a transaction branch. (The TM's set of RM switches, described in Section 4.3 on page 21 tells the TM which RMs are associated with it.) The TM calls all these RMs with *xa_start*( ), *xa_end*( ), and *xa_prepare*( ), although an RM that is not active in a branch need not participate further (see Section 2.3.2 on page 8). A technique to reduce overhead for infrequently-used RMs is discussed below.

Dynamic Registration

Certain RMs, especially those involved in relatively few global transactions, may ask the TM to assume they are *not* involved in a transaction. These RMs must register with the TM before they do application work, to see whether the work is part of a global transaction. The TM never calls these RMs with any form of *xa_start*( ). An RM declares dynamic registration in its switch (see Section 4.3 on page 21). An RM can make this declaration only on its own behalf, and doing so does not change the TM's behaviour with respect to other RMs.

When an AP requests work from such an RM, before doing any work, the RM contacts the TM by calling *ax_reg*( ). The RM must call *ax_reg*( ) from the same thread of control that the AP would use if it called *ax_reg*( ) directly. The TM returns to the RM the appropriate XID if the AP is in a global transaction.

If the thread ends its involvement in the transaction branch (using *xa_end*( )), then the RM must re-register (using *ax_reg*( )) with the TM if the AP calls it for additional work in the global transaction. If the RM does not resume its participation, then the TM does not call the RM again for that branch until the TM completes the branch.

If the RM calls *ax_reg*( ) and the AP is not in a global transaction, the TM informs the RM, and remembers that the RM is doing work outside any global transaction. In this case, when the AP completes its work with the RM, the RM must notify the TM by calling *ax_unreg*( ). The RM must call *ax_unreg*( ) from the same thread of control from which it called *ax_reg*( ). Until then — that is, as long as the AP thread involves the RM outside a global transaction — the TM

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

neither lets the AP start a global transaction, nor lets any RM register through the same thread to participate in one.

3.4 Branch Completion

A TM calls *xa_prepare*( ) to ask the RM to prepare to commit a transaction branch. The RM places any resources it holds in a state such that it can either make any changes permanent if the TM subsequently calls *xa_commit*( ), or nullify any changes if the TM calls *xa_rollback*( ). An affirmative return from *xa_prepare*( ) guarantees that a subsequent *xa_commit*( ) or *xa_rollback*( ) succeeds, even if the RM experiences a failure after responding to *xa_prepare*( ).

A TM calls *xa_commit*( ) to direct the RM to commit a transaction branch. The RM applies permanently any changes it has made to shared resources, and releases any resources it held on behalf of the branch. A TM calls *xa_rollback*( ) to ask the RM to roll back a branch. The RM undoes any changes that it applied to shared resources, and releases any resources it held.

Before a TM can call *xa_prepare*( ) for a transaction branch, all associations must be completely ended with *xa_end*( ) (see Section 3.3 on page 14). Any thread can then initiate branch completion. That is, the TM may supervise branch completion with a separate thread from the AP threads that did work on behalf of the global transaction.

Optimisations

This section describes the use of *xa_** routines in the standard two-phase commit protocol. See Section 2.3.2 on page 8 for other permissible sequences of these calls.

Heuristic Decision

The X/Open DTP model lets RMs complete transaction branches heuristically (see Section 2.3.3 on page 9). The RM cannot discard its knowledge of such a branch until the TM permits this by calling *xa_forget*( ) for each branch.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.5 Synchronous, Non-blocking, and Asynchronous Modes

Synchronous

The *xa_*()* functions typically operate synchronously; control does not return to the caller until the operation is complete. Some routines, notably *xa_start*() (see Section 3.3 on page 14) may block the calling thread.

Two other calling modes help the TM schedule its work when dealing with several RMs:

Non-blocking

Certain *xa_*()* calls direct the RM to operate synchronously with the caller but without blocking it. If the RM cannot complete the call without blocking, it reports this immediately.

Asynchronous

Most *xa_*()* routines have a form by which the caller requests asynchrony. Asynchronous calls should return immediately. The caller can subsequently call *xa_complete*() to test the asynchronous operation for completion.

A TM might give an RM an asynchronous request (particularly a request to prepare to commit a transaction branch) so that the TM could do other work in the meantime. Within the same thread of control, a TM cannot use asynchrony to give additional work to the same RM for the same branch; the only *xa_*()* call a TM can give to the RM for the same branch is *xa_complete*() to test that operation's completion. However, for the branch-completion routines: *xa_commit*(), *xa_prepare*(), and *xa_rollback*(), and for *xa_forget*(), the TM may issue multiple commands to the same RM from within the same thread of control. Each of these commands must be for a different branch.

3.6 Failure Recovery

A TM must ensure orderly completion of all transaction branches. A TM calls *xa_recover*() during failure recovery to get a list of all branches that an RM has prepared or heuristically completed.

Unilateral RM Action

An RM can mark a transaction branch as rollback-only any time except after a successful prepare. A TM detects this when the RM returns a rollback-only return code. An RM can also unilaterally roll back and forget a branch any time except after a successful prepare. A TM detects this when a subsequent call indicates that the RM does not recognise the XID. The former technique gives the TM more information.

If a thread of control terminates, an RM must dissociate and roll back any associated transaction branch. If an RM experiences machine failure or termination, it must also unilaterally roll back all branches that have not successfully prepared.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Chapter 4: The "xa.h" Header

This chapter specifies structure definitions, flags, and error codes to which conforming products must adhere. It also declares the routines by which RMs call a TM. (Entry points to an RM are contained in the RM's switch; see Section 4.3 on page 21.) This is the minimum content of an include file called **"xa.h"**. Fully standardising this information lets RMs be written independently of the TMs that use them. It also lets users interchange TMs and RMs without recompiling.

Appendix A contains an **"xa.h"** header file with **#define** statements suitable for ANSI C (see the referenced C standard) and Common Usage C implementations. This chapter contains excerpts from the ANSI C code in **"xa.h"**. The synopses in Chapter 5 also use ANSI C.

4.1 Naming Conventions

The XA interface uses certain naming conventions to name its functions, flags, and return codes. All names that appear in **"xa.h"** are part of the XA name space. This section describes the XA naming conventions.

- The negative (error) codes returned by the *xa_*()* routines all begin with XAER_. The non-negative return codes all begin with XA_.
- The names of all TM functions that RMs call begin with *ax_* (for example, *ax_reg*). Their negative (error) return codes all begin with TMER_. Their non-negative return codes all begin with TM_.
- Names of flags passed to XA routines, and of flags in the RM switch, begin with TM.

4.2 Transaction Identification

The **"xa.h"** header defines a public structure called an XID to identify a transaction branch. RMs and TMs both use the XID structure. This lets an RM work with several TMs without recompilation.

The XID structure is specified in the C code below in **struct xid_t**. The XID contains a format identifier, two length fields, and a data field. The data field comprises at most two contiguous components: a global transaction identifier (*gtrid*) and a branch qualifier (*bqual*).

The *gtrid_length* element specifies the number of bytes (1-64) that constitute *gtrid*, starting at the first byte of the data element (that is, at *data*[0]). The *bqual_length* element specifies the number of bytes (1-64) that constitute *bqual*, starting at the first byte after *gtrid* (that is, at *data*[*gtrid_length*]). Neither component in *data* is null-terminated. The TM need not initialise any unused bytes in *data*.

Although **"xa.h"** constrains the length and byte alignment of the data element within an XID, it does not specify the data's contents. The only requirement is that both *gtrid* and *bqual*, taken together, must be globally unique. The recommended way of achieving global uniqueness is to use the naming rules specified for OSI CCR atomic action identifiers (see the referenced

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

OSI CCR standards). If OSI CCR naming is used, then the XID's *formatID* element should be set to 0; if some other format is used, then the *formatID* element should be greater than 0. A value of –1 in *formatID* means that the XID is null.

The RM must be able to map the XID to the recoverable work it did for the corresponding branch. RMs may perform bitwise comparisons on the data components of an XID for the lengths specified in the XID structure. Most XA routines pass a pointer to the XID. These pointers are valid only for the duration of the call. If the RM needs to refer to the XID after it returns from the call, it must make a local copy before returning.

```
/*
 * Transaction branch identification: XID and NULLXID:
 */
#define XIDDATASIZE  128          /* size in bytes */
#define MAXGTRIDSIZE 64           /* maximum size in bytes of gtrid */
#define MAXBQUALSIZE 64           /* maximum size in bytes of bqual */
struct xid_t {
   long formatID;     /* format identifier */
   long gtrid_length; /* value 1-64 */
   long bqual_length; /* value 1-64 */
   char data[XIDDATASIZE];
   };
typedef struct xid_t XID;
/*
 * A value of -1 in formatID means that the XID is null.
 */
/*
 * Declarations of routines by which RMs call TMs:
 */
extern int ax_reg(int, XID *, long);
extern int ax_unreg(int, long);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.3 Resource Manager Switch

The TM administrator can add or remove an RM from the DTP system by simply controlling the set of RMs linked to executable modules — how this is achieved for a specific system is implementation defined (see Chapter 7). Each RM must provide a *switch* that gives the TM access to the RM's *xa_*()* routines. This lets the administrator change the set of RMs linked with an executable module without having to recompile the application. A different set of RMs and their switches may be linked into each separate application executable module in the DTP system. Several instances of an RM can share the RM's switch structure.

An RM's switch uses a structure called **xa_switch_t**. The switch contains the RM's name, non-null pointers to the RM's entry points, a flag, and a version word. The flags tell whether the RM uses dynamic registration (see Section 3.3.1 on page 15), whether the RM operates asynchronously (see Section 3.5 on page 17), and whether the RM supports the migration of associations (see Section 3.3 on page 14). Section 4.4 on page 22 defines constants used as these flags. The RM cannot change these declarations during the operation of the DTP system.

```
/*
 * XA Switch Data Structure
 */
#define RMNAMESZ 32             /* length of resource manager name, */
                                /* including the null terminator */
#define MAXINFOSIZE 256         /* maximum size in bytes of xa_info strings,*/
                                /* including the null terminator */
struct xa_switch_t {
   char name[RMNAMESZ];         /* name of resource manager */
   long flags;                  /* options specific to the resource manager */
   long version;                /* must be 0 */
   int (*xa_open_entry)(char *, int, long); /* xa_open function pointer */
   int (*xa_close_entry)(char *, int, long);/* xa_close function pointer */
   int (*xa_start_entry)(XID *, int, long); /* xa_start function pointer */
   int (*xa_end_entry)(XID *, int, long);   /* xa_end function pointer */
   int (*xa_rollback_entry)(XID *, int, long);
                                            /* xa_rollback function pointer*/
   int (*xa_prepare_entry)(XID *, int, long);
                                            /* xa_prepare function pointer */
   int (*xa_commit_entry)(XID *, int, long);/* xa_commit function pointer */
   int (*xa_recover_entry)(XID *, long, int, long);
                                            /* xa_recover function pointer */
   int (*xa_forget_entry)(XID *, int, long);/* xa_forget function pointer */
   int (*xa_complete_entry)(int *, int *, int, long);
                                            /* xa_complete function pointer*/
};
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.4 Flag Definitions

The XA interface uses the flag definitions. For a TM to work with different RMs without change or recompilation, each RM uses these flags, defined in the **"xa.h"** header.

The **"xa.h"** header defines a constant, TMNOFLAGS, for use in situations where no other flags are specified. An RM that does not use any flags to specify special features in its switch (see Section 4.3 on page 21) should specify TMNOFLAGS. In addition, TMs and RMs should use the same TMNOFLAGS constant as the flag argument in any *xa_*()* or *ax_*()* call in which they do not use explicit options.

Flag definitions for the XA interface are as follows:

```
/*
 * Flag definitions for the RM switch
 */
#define TMNOFLAGS      0x00000000L   /* no resource manager features
                                        selected */
#define TMREGISTER     0x00000001L   /* resource manager dynamically
                                        registers */
#define TMNOMIGRATE    0x00000002L   /* resource manager does not support
                                        association migration */
#define TMUSEASYNC     0x00000004L   /* resource manager supports
                                        asynchronous operations */
/*
 * Flag definitions for xa_ and ax_ routines
 */
/* use TMNOFLAGS, defined above, when not specifying other flags */
#define TMASYNC        0x80000000L   /* perform routine asynchronously */
#define TMONEPHASE     0x40000000L   /* caller is using one-phase commit
                                        optimisation */
#define TMFAIL         0x20000000L   /* dissociates caller and marks
                                        transaction branch rollback-only */
#define TMNOWAIT       0x10000000L   /* return if blocking condition exists */
#define TMRESUME       0x08000000L   /* caller is resuming association
                                        with suspended transaction branch */
#define TMSUCCESS      0x04000000L   /* dissociate caller from transaction
                                        branch */
#define TMSUSPEND      0x02000000L   /* caller is suspending, not ending,
                                        association */
#define TMSTARTRSCAN   0x01000000L   /* start a recovery scan */
#define TMENDRSCAN     0x00800000L   /* end a recovery scan */
#define TMMULTIPLE     0x00400000L   /* wait for any asynchronous operation */
#define TMJOIN         0x00200000L   /* caller is joining existing transaction
                                        branch */
#define TMMIGRATE      0x00100000L   /* caller intends to perform migration */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The semantics of a flag setting when used on a particular function call are described on the manual page of the call as follows:

Table 4-1 Flags used in Particular Function Calls

Flag	Function
TMASYNC	*xa_close*( ) on page 32 *xa_commit*( ) on page 34 *xa_end*( ) on page 38 *xa_forget*( ) on page 41 *xa_open*( ) on page 43 *xa_prepare*( ) on page 45 *xa_rollback*( ) on page 50 *xa_start*( ) on page 53
TMENDRSCAN	*xa_recover*( ) on page 48
TMFAIL	*xa_end*( ) on page 38
TMJOIN	*xa_start*( ) on page 53
TMMIGRATE	*xa_end*( ) on page 38
TMMULTIPLE	*xa_complete*( ) on page 37
TMNOWAIT	*xa_commit*( ) on page 34 *xa_complete*( ) on page 37 *xa_start*( ) on page 53
TMONEPHASE	*xa_commit*( ) on page 34
TMRESUME	*xa_start*( ) on page 53
TMSTARTRSCAN	*xa_recover*( ) on page 48
TMSUCCESS	*xa_end*( ) on page 38
TMSUSPEND	*xa_end*( ) on page 38

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.5 Return Codes

As with flag definitions, all TMs and RMs must ensure interchangeability by using these return codes, defined in the "**xa.h**" header.

```
/*
 * ax_() return codes (transaction manager reports to resource manager)
 */
#define TM_JOIN      2    /* caller shall join work of existing transaction
                             branch */
#define TM_RESUME    1    /* caller is resuming association
                             with suspended transaction branch */
#define TM_OK        0    /* normal execution */
#define TMER_TMERR   -1   /* an error occurred in the
                             transaction manager */
#define TMER_INVAL   -2   /* invalid arguments were given */
#define TMER_PROTO   -3   /* routine invoked in an improper context */

/*
 * xa_() return codes (resource manager reports to transaction manager)
 */
#define XA_RBBASE       100           /* the inclusive lower bound of the
                                         rollback codes */
#define XA_RBROLLBACK   XA_RBBASE     /* the rollback was caused by an
                                         unspecified reason */
#define XA_RBCOMMFAIL   XA_RBBASE+1   /* the rollback was caused by a
                                         communication failure */
#define XA_RBDEADLOCK   XA_RBBASE+2   /* a deadlock was detected */
#define XA_RBINTEGRITY XA_RBBASE+3    /* a condition that violates the
                                         integrity of the resources
                                         was detected */
#define XA_RBOTHER      XA_RBBASE+4   /* the resource manager rolled back the
                                         transaction branch for a reason not on
                                         this list */
#define XA_RBPROTO      XA_RBBASE+5   /* a protocol error occurred in the
                                         resource manager */
#define XA_RBTIMEOUT    XA_RBBASE+6   /* a transaction branch took too long*/
#define XA_RBTRANSIENT XA_RBBASE+7    /* may retry the transaction branch */
#define XA_RBEND        XA_RBTRANSIENT /* the inclusive upper bound of the
                                          rollback codes */

#define XA_NOMIGRATE   9   /* resumption must occur where
                              suspension occurred */
#define XA_HEURHAZ     8   /* the transaction branch may have
                              been heuristically completed */
#define XA_HEURCOM     7   /* the transaction branch has been
                              heuristically committed */
#define XA_HEURRB      6   /* the transaction branch has been
                              heuristically rolled back */
#define XA_HEURMIX     5   /* the transaction branch has been
                              heuristically committed and rolled back */
#define XA_RETRY       4   /* routine returned with no effect and
                              may be reissued */
#define XA_RDONLY      3   /* the transaction branch was read-only and
                              has been committed */
#define XA_OK          0   /* normal execution */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
#define XAER_ASYNC     -2   /* asynchronous operation already outstanding */
#define XAER_RMERR     -3   /* a resource manager error occurred in the
                            transaction branch */
#define XAER_NOTA      -4   /* the XID is not valid */
#define XAER_INVAL     -5   /* invalid arguments were given */
#define XAER_PROTO     -6   /* routine invoked in an improper context */
#define XAER_RMFAIL    -7   /* resource manager unavailable */
#define XAER_DUPID     -8   /* the XID already exists */
#define XAER_OUTSIDE   -9   /* resource manager doing work outside */
                            /* global transaction */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Chapter 5: Reference Manual Pages

This chapter describes the interfaces to the XA service set. Reference manual pages appear in alphabetical order, for each service in the XA interface. The *ax_*()* routines are provided by a TM for use by RMs. The *xa_*()* routines are provided by each RM for use by the TM.

The symbolic constants and error names are described in the **"xa.h"** header (see Chapter 4). The state tables referred to in the reference manual pages appear in Chapter 6.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ax_reg — dynamically register a resource manager with a transaction manager

SYNOPSIS

```
#include "xa.h"

int
ax_reg(int rmid, XID *xid, long flags)
```

DESCRIPTION

A resource manager calls *ax_reg*( ) to inform a transaction manager that it is about to perform work on behalf of an application thread of control. The transaction manager, in turn, replies to the resource manager with an indication of whether or not that work should be performed on behalf of a transaction branch. If the transaction manager determines that the calling thread of control is involved in a branch, upon successful return, the space to which *xid* points contains a valid XID. If the resource manager's work is outside any global transaction, the space to which *xid* points contains NULLXID. The caller is responsible for allocating the space to which *xid* points.

A resource manager must call this function from the same thread of control from which the application accesses the resource manager. A resource manager taking advantage of this facility must have TMREGISTER set in the *flags* element of its **xa_switch_t** structure (see Chapter 4). Moreover, *ax_reg*( ) returns failure, [TMER_TMERR], when issued by a resource manager that has not set TMREGISTER.

When the resource manager calls *ax_reg*( ) for a new thread of control association (that is, when [TM_RESUME] is not returned; see below), the transaction manager may generate a unique branch qualifier within the returned XID.

If the resource manager is resuming work on a suspended transaction branch, it informs the resource manager of this by returning [TM_RESUME]. When [TM_RESUME] is returned, *xid* points to the same XID that was passed to the *xa_end*( ) call that suspended the association. If the resource manager receives [TM_RESUME] and does not recognise the transaction identified by **xid*, it must return a failure indication to the application. Failure to do so will result in indeterminate behaviour.

If the transaction manager generated a new branch qualifier within the returned XID, this thread is loosely-coupled in relation to the other threads in this same global transaction. That is, the resource manager may treat this thread's work as a separate transaction with respect to its isolation policies.

If the transaction manager elects to reuse within **xid* a branch qualifier previously given to the resource manager, it informs the resource manager of this by returning [TM_JOIN]. This directs the resource manager to join the work of the existing transaction branch. If the resource manager receives [TM_JOIN] and does not recognise the transaction identified by **xid*, it must return a failure indication to the application. Failure to do so will result in indeterminate behaviour. If the transaction manager reuses a branch qualifier within the returned XID, this thread is tightly-coupled to the other threads in the same transaction branch. A resource manager must guarantee that tightly-coupled threads are treated as a single entity with respect to its isolation policies and that no resource deadlock can occur within the transaction branch among these tightly-coupled threads. Failure to do so will result in indeterminate behaviour.

The implications of dynamically registering are as follows: when a thread of control begins working on behalf of a transaction branch, the transaction manager calls *xa_start*( ) for all resource managers known to the thread except those having TMREGISTER set in their **xa_switch_t** structure. Thus, those resource managers with this flag set must explicitly join a branch with *ax_reg*( ). Secondly, when a thread of control is working on behalf of a branch, a

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

transaction manager calls *xa_end*( ) for all resource managers known to the thread that either do not have TMREGISTER set in their **xa_switch_t** structure or have dynamically registered with *ax_reg*( ).

The function's first argument, *rmid*, is the integer that the resource manager received when the transaction manager called *xa_open*( ). It identifies the resource manager in the thread of control.

The function's last argument, *flags*, is reserved for future use and must be set to TMNOFLAGS.

RETURN VALUE

The function *ax_reg*( ) has the following return values:

[TM_JOIN]
The resource manager shall join the work of an existing transaction branch. The resource manager should make available enough transaction context so that tightly-coupled threads are not susceptible to resource deadlock within the branch. If the resource manager does not recognise the transaction identified by **xid*, it must return a failure indication to the application. Failure to do so will result in indeterminate behaviour.

[TM_RESUME]
The resource manager shall resume work on a previously-suspended transaction branch. Failure to do so will result in indeterminate behaviour. The resource manager should make available at least the transaction context that is specific to the resource manager, present at the time of the suspend, as if the thread had effectively never been suspended, except that other threads in the global transaction may have affected this context.

If the resource manager does not recognise the transaction identified by **xid*, it must return a failure indication to the application. Failure to do so will result in indeterminate behaviour. If the resource manager allows an association to be resumed in a different thread from the one that suspended the work, and the transaction manager expressed its intention to migrate the association (via the TMMIGRATE flag on *xa_end*( )), the current thread may be different from the one that suspended the work. Otherwise, the current thread must be the same, or the resource manager must return a failure indication to the application; failure to do so will result in indeterminate behaviour.

If **xid* contains a reused branch qualifier, and the transaction manager has multiple outstanding suspended thread associations for **xid*, the following rules apply:

- The transaction manager can have only one of them outstanding at any time with TMMIGRATE set in *flags*.
- Moreover, the transaction manager cannot resume this association in a thread that currently has a non-migratable suspended association.

These rules prevent ambiguity as to which context is restored.

[TM_OK]
Normal execution.

[TMER_TMERR]
The transaction manager encountered an error in registering the resource manager.

[TMER_INVAL]
Invalid arguments were specified.

[TMER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SEE ALSO

ax_unreg( ), *xa_end*( ), *xa_open*( ), *xa_start*( ).

WARNINGS

If *xid* does not point to a buffer that is at least as large as the size of an XID, *ax_reg*( ) may overwrite the caller's data space. In addition, the buffer must be properly aligned (on a long word boundary) in the event that structure assignments are performed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

ax_unreg — dynamically unregister a resource manager with a transaction manager

SYNOPSIS

```
#include "xa.h"

int
ax_unreg(int rmid, long flags)
```

DESCRIPTION

A resource manager calls *ax_unreg*( ) to inform a transaction manager that it has completed work, outside any global transaction, that it began after receiving the NULLXID from *ax_reg*( ). In addition, the resource manager is informing the transaction manager that the accessing thread of control is free to participate (from the resource manager's perspective) in a global transaction. So long as any resource manager in a thread of control is registered with a transaction manager and is performing work outside any global transaction, that application thread cannot participate in a global transaction.

A resource manager must call this function from the same thread of control that originally called *ax_reg*( ). A resource manager taking advantage of this facility must have TMREGISTER set in the *flags* element of its **xa_switch_t** structure (see Chapter 4). Moreover, *ax_unreg*( ) returns failure [TMER_TMERR] when issued by a resource manager that has not set TMREGISTER.

The function's first argument, *rmid*, is the integer that the resource manager received when the transaction manager called *xa_open*( ). It identifies the resource manager in the thread of control.

The function's last argument, *flags*, is reserved for future use and must be set to TMNOFLAGS.

RETURN VALUE

ax_unreg( ) has the following return values:

[TM_OK]
Normal execution.

[TMER_TMERR]
The transaction manager encountered an error in unregistering the resource manager.

[TMER_INVAL]
Invalid arguments were specified.

[TMER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

SEE ALSO

ax_reg( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

xa_close — close a resource manager

SYNOPSIS

```
#include "xa.h"

int
xa_close(char *xa_info, int rmid, long flags)
```

DESCRIPTION

A transaction manager typically calls *xa_close*( ) to close a currently open resource manager in the calling thread of control. Once closed, the resource manager cannot participate in global transactions on behalf of the calling thread until it is re-opened.

The argument *xa_info* points to a null-terminated character string that may contain instance-specific information for the resource manager. The maximum length of this string is 256 bytes (including the null terminator). The argument *xa_info* may point to an empty string if the resource manager does not require instance-specific information to be available. The argument *xa_info* must not be a null pointer.

A transaction manager must call this function from the same thread of control that accesses the resource manager. In addition, attempts to close a resource manager that is already closed have no effect and return success, [XA_OK].

It is an error, [XAER_PROTO], for a transaction manager to call *xa_close*( ) within a thread of control that is associated with a transaction branch (that is, the transaction manager must call *xa_end*( ) before calling *xa_close*( )). In addition, if a transaction manager calls *xa_close*( ) while an asynchronous operation is pending at a resource manager, an error, [XAER_PROTO], is returned.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

The function's last argument, *flags* must be set to one of the following values:

TMASYNC
: This flag indicates that *xa_close*( ) shall be performed asynchronously. Upon success, the function returns a positive value (called a handle) that the caller can use as an argument to *xa_complete*( ) to wait for the operation to complete. If the calling thread of control already has an asynchronous operation pending at the same resource manager, this function fails, returning [XAER_ASYNC].

TMNOFLAGS
: This flag must be used when no other flags are set in *flags*.

RETURN VALUE

The function *xa_close*( ) has the following return values:

[XA_OK]
: Normal execution.

[XAER_ASYNC]
: TMASYNC was set in *flags*, and either the maximum number of outstanding asynchronous operations has been exceeded, or TMUSEASYNC is not set in the *flags* element of the resource manager's **xa_switch_t** structure.

[XAER_RMERR]
: An error occurred when closing the resource.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[XAER_INVAL]
Invalid arguments were specified.

[XAER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

The function returns a positive value upon success if the caller set TMASYNC (see above) in *flags*.

SEE ALSO

xa_complete( ), *xa_end*( ), *xa_open*( ).

WARNINGS

From the resource manager's perspective, the pointer *xa_info* is valid only for the duration of the call to *xa_close*( ). That is, once the function completes, either synchronously or asynchronously, the transaction manager is allowed to invalidate where *xa_info* points. Resource managers are encouraged to use private copies of **xa_info* after the function completes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

xa_commit — commit work done on behalf of a transaction branch

SYNOPSIS

```
#include "xa.h"

int
xa_commit(XID *xid, int rmid, long flags)
```

DESCRIPTION

A transaction manager calls *xa_commit*( ) to commit the transaction identified by **xid*. Any changes made to resources held during the transaction branch are made permanent. A transaction manager may call this function from any thread of control. All associations for the transaction identified by **xid* must have been ended by using *xa_end*( ) with TMSUCCESS set in *flags*.

If a resource manager already completed the work associated with **xid* heuristically, this function merely reports how the resource manager completed the transaction branch. A resource manager shall not forget about a heuristically completed branch until the transaction manager calls *xa_forget*( ). Otherwise, indeterminate behaviour will result.

A transaction manager must issue a separate *xa_commit*( ) for each transaction branch that accessed the resource manager.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

Following are the valid settings of *flags* (note that at most one of TMNOWAIT and TMASYNC may be set):

TMNOWAIT
: When this flag is set and a blocking condition exists, *xa_commit*( ) returns [XA_RETRY] and does not commit the transaction branch (that is, the call has no effect). The function *xa_commit*( ) must be called at a later time to commit the branch. TMNOWAIT is ignored if TMONEPHASE is set.

TMASYNC
: This flag indicates that *xa_commit*( ) shall be performed asynchronously. Upon success, the function returns a positive value (called a handle) that the caller can use as an argument to *xa_complete*( ) to wait for the operation to complete. If the calling thread of control already has an asynchronous operation pending at the same resource manager for the same XID, this function fails, returning [XAER_ASYNC].

TMONEPHASE
: The transaction manager must set this flag if it is using the one-phase commit optimisation for the specified transaction branch.

TMNOFLAGS
: This flag must be used when no other flags are set in *flags*.

RETURN VALUE

The function *xa_commit*( ) has the following return values:

[XA_HEURHAZ]
: Due to some failure, the work done on behalf of the specified transaction branch may have been heuristically completed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[XA_HEURCOM]
Due to a heuristic decision, the work done on behalf of the specified transaction branch was committed.

[XA_HEURRB]
Due to a heuristic decision, the work done on behalf of the specified transaction branch was rolled back.

[XA_HEURMIX]
Due to a heuristic decision, the work done on behalf of the specified transaction branch was partially committed and partially rolled back.

[XA_RETRY]
The resource manager is not able to commit the transaction branch at this time. This value may be returned when a blocking condition exists and TMNOWAIT was set. Note, however, that this value may also be returned even when TMNOWAIT is not set (for example, if the necessary stable storage is currently unavailable). This value cannot be returned if TMONEPHASE is set in *flags*. All resources held on behalf of the transaction identified by **xid* remain in a prepared state until commitment is possible. The transaction manager should reissue *xa_commit*( ) at a later time.

[XA_OK]
Normal execution.

[XA_RB*]
The resource manager did not commit the work done on behalf of the transaction branch. Upon return, the resource manager has rolled back the branch's work and has released all held resources. These values may be returned only if TMONEPHASE is set in *flags*:

[XA_RBROLLBACK]
The resource manager rolled back the transaction branch for an unspecified reason.

[XA_RBCOMMFAIL]
A communication failure occurred within the resource manager.

[XA_RBDEADLOCK]
The resource manager detected a deadlock.

[XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.

[XA_RBOTHER]
The resource manager rolled back the transaction branch for a reason not on this list.

[XA_RBPROTO]
A protocol error occurred within the resource manager.

[XA_RBTIMEOUT]
The work represented by this transaction branch took too long.

[XA_RBTRANSIENT]
The resource manager detected a transient error.

[XAER_ASYNC]
TMASYNC was set in *flags*, and either the maximum number of outstanding asynchronous operations has been exceeded, or TMUSEASYNC is not set in the *flags* element of the resource manager's **xa_switch_t** structure.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[XAER_RMERR]
An error occurred in committing the work performed on behalf of the transaction branch and the branch's work has been rolled back. Note that returning this error signals a catastrophic event to a transaction manager since other resource managers may successfully commit their work on behalf of this branch. This error shall be returned only when a resource manager concludes that it can never commit the branch and that it cannot hold the branch's resources in a prepared state. Otherwise, [XA_RETRY] should be returned.

[XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.

[XAER_NOTA]
The specified XID is not known by the resource manager.

[XAER_INVAL]
Invalid arguments were specified.

[XAER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

The function returns a positive value upon success if the caller set TMASYNC (see above) in *flags*.

SEE ALSO

xa_complete( ), *xa_forget*( ), *xa_open*( ), *xa_prepare*( ), *xa_rollback*( ).

WARNINGS

From the resource manager's perspective, the pointer *xid* is valid only for the duration of the call to *xa_commit*( ). That is, once the function completes, either synchronously or asynchronously, the transaction manager is allowed to invalidate where *xid* points. Resource managers are encouraged to use private copies of **xid* after the function completes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

xa_complete — wait for an asynchronous operation to complete

SYNOPSIS

```
#include "xa.h"

int
xa_complete(int *handle, int *retval, int rmid, long flags)
```

DESCRIPTION

A transaction manager calls *xa_complete*( ) to wait for the completion of an asynchronous operation. By default, this function waits for the operation pointed to by *handle* to complete. The argument **handle* must have previously been returned by a function that had TMASYNC set. In addition, a transaction manager must call *xa_complete*( ) from the same thread of control that received **handle*.

Upon successful return, [XA_OK], *retval* points to the return value of the asynchronous operation and **handle* is no longer valid. If *xa_complete*( ) returns any other value, **handle*, **retval*, and any outstanding asynchronous operation are not affected. The caller is responsible for allocating the space to which *handle* and *retval* point.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

Following are the valid settings of *flags*:

TMMULTIPLE
: When this flag is set, *xa_complete*( ) tests for the completion of any outstanding asynchronous operation. The argument **handle* is ignored if TMMULTIPLE is set.

TMNOWAIT
: When this flag is set, *xa_complete*( ) tests for the completion of an operation without blocking. That is, if the operation denoted by **handle* (or any operation, if TMMULTIPLE is also set) has not completed, *xa_complete*( ) returns [XA_RETRY] and does not wait for the operation to complete.

TMNOFLAGS
: This flag must be used when no other flags are set in *flags*.

RETURN VALUE

The function *xa_complete*( ) has the following return values:

[XA_RETRY]
: TMNOWAIT was set in *flags* and no asynchronous operation has completed.

[XA_OK]
: Normal execution.

[XAER_PROTO]
: The routine was invoked in an improper context. See Chapter 6 for details.

[XAER_INVAL]
: Invalid arguments were specified.

SEE ALSO

xa_close( ), *xa_commit*( ), *xa_end*( ), *xa_forget*( ), *xa_open*( ), *xa_prepare*( ), *xa_rollback*( ), *xa_start*( ).

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

xa_end — end work performed on behalf of a transaction branch

SYNOPSIS

```
#include "xa.h"

int
xa_end(XID *xid, int rmid, long flags)
```

DESCRIPTION

A transaction manager calls *xa_end*( ) when a thread of control finishes, or needs to suspend work on, a transaction branch. This occurs when the application completes a portion of its work, either partially or in its entirety (for example, before blocking on some event in order to let other threads of control work on the branch). When *xa_end*( ) successfully returns, the calling thread of control is no longer actively associated with the branch but the branch still exists. A transaction manager must call this function from the same thread of control that accesses the resource manager.

A transaction manager always calls *xa_end*( ) for those resource managers that do not have TMREGISTER set in the *flags* element of their **xa_switch_t** structure. Unlike *xa_start*( ), *xa_end*( ) is also issued to those resource managers that have previously registered with *ax_reg*( ). After the transaction manager calls *xa_end*( ), it should no longer consider the calling thread associated with that resource manager (although it must consider the resource manager part of the transaction branch when it prepares the branch.) Thus, a resource manager that dynamically registers must re-register after an *xa_end*( ) that suspends its association (that is, after an *xa_end*( ) with TMSUSPEND set in *flags*) but before the application thread of control continues to access the resource manager.

The first argument, *xid*, is a pointer to an XID. The argument *xid* must point to the same XID that was either passed to the *xa_start*( ) call or returned from the *ax_reg*( ) call that established the thread's association; otherwise, an error, [XAER_NOTA], is returned.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

Following are the valid settings of *flags* (note that one and only one of TMSUSPEND, TMSUCCESS, or TMFAIL must be set):

TMSUSPEND

Suspend a transaction branch on behalf of the calling thread of control. For a resource manager that allows multiple threads of control, but only one at a time working on a specific branch, it might choose to allow another thread of control to work on the branch at this point. If this flag is not accompanied by the TMMIGRATE flag, the transaction manager must resume or end the suspended association in the current thread. TMSUSPEND cannot be used in conjunction with either TMSUCCESS or TMFAIL.

TMMIGRATE

The transaction manager intends (but is not required) to resume the association in a thread different from the calling one. This flag may be used only in conjunction with TMSUSPEND and only with a resource manager that does not have TMNOMIGRATE set in the flags element of its **xa_switch_t** structure. Setting TMMIGRATE in flags, while another thread's association for **xid* is currently suspended with TMMIGRATE, makes *xa_end*( ) fail, returning [XAER_PROTO]. This is because a transaction manager can have at any given time at most one suspended thread association migrating for a particular transaction branch. If this flag is not used, a transaction manager is required to resume or end the association in the current thread.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TMSUCCESS
The portion of work has succeeded. This flag cannot be used in conjunction with either TMSUSPEND or TMFAIL.

TMFAIL
The portion of work has failed. A resource manager might choose to mark a transaction branch as rollback-only at this point. In fact, a transaction manager does so for the global transaction. If a resource manager chooses to do so also, *xa_end*( ) returns one of the [XA_RB*] values. TMFAIL cannot be used in conjunction with either TMSUSPEND or TMSUCCESS.

TMASYNC
This flag indicates that *xa_end*( ) shall be performed asynchronously. Upon success, the function returns a positive value (called a handle) that the caller can use as an argument to *xa_complete*( ) to wait for the operation to complete. If the calling thread of control already has an asynchronous operation pending at the same resource manager for the same XID, this function fails, returning [XAER_ASYNC].

RETURN VALUE

The function *xa_end*( ) has the following return values:

[XA_NOMIGRATE]
The resource manager was unable to prepare the transaction context for migration. However, the resource manager has suspended the association. The transaction manager can resume the association only in the current thread. This code may be returned only when both TMSUSPEND and TMMIGRATE are set in *flags*. A resource manager that sets TMNOMIGRATE in the *flags* element of its **xa_switch_t** structure need not return [XA_NOMIGRATE].

[XA_OK]
Normal execution.

[XA_RB*]
The resource manager has dissociated the transaction branch from the thread of control and has marked rollback-only the work performed on behalf of the transaction identified by **xid*. The following values may be returned regardless of the setting of *flags*:

[XA_RBROLLBACK]
The resource manager marked the transaction branch rollback-only for an unspecified reason.

[XA_RBCOMMFAIL]
A communication failure occurred within the resource manager.

[XA_RBDEADLOCK]
The resource manager detected a deadlock.

[XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.

[XA_RBOTHER]
The resource manager marked the transaction branch rollback-only for a reason not on this list.

[XA_RBPROTO]
A protocol error occurred within the resource manager.

[XA_RBTIMEOUT]
The work represented by this transaction branch took too long.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[XA_RBTRANSIENT]
The resource manager detected a transient error.

[XAER_ASYNC]
TMASYNC was set in *flags*, and either the maximum number of outstanding asynchronous operations has been exceeded, or TMUSEASYNC is not set in the *flags* element of the resource manager's **xa_switch_t** structure.

[XAER_RMERR]
An error occurred in dissociating the transaction branch from the thread of control.

[XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.

[XAER_NOTA]
The specified XID is not known by the resource manager.

[XAER_INVAL]
Invalid arguments were specified.

[XAER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

The function returns a positive value upon success if the caller set TMASYNC (see above) in *flags*.

SEE ALSO

ax_reg( ), *xa_complete*( ), *xa_open*( ), *xa_start*( ).

WARNINGS

From the resource manager's perspective, the pointer *xid* is valid only for the duration of the call to *xa_end*( ). That is, once the function completes, either synchronously or asynchronously, the transaction manager is allowed to invalidate where *xid* points. Resource managers are encouraged to use private copies of **xid* after the function completes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

xa_forget — forget about a heuristically completed transaction branch

SYNOPSIS

```
#include "xa.h"

int
xa_forget(XID *xid, int rmid, long flags)
```

DESCRIPTION

A resource manager that heuristically completes work done on behalf of a transaction branch must keep track of that branch along with the decision (that is, heuristically committed, rolled back, or mixed) until told otherwise. The transaction manager calls *xa_forget*( ) to permit the resource manager to erase its knowledge of the transaction identified by **xid*. Upon successful return, [XA_OK], **xid* is no longer valid (from the resource manager's point of view). A transaction manager may call this function from any thread of control.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

The function's last argument, *flags*, must be set to one of the following values:

TMASYNC
: This flag indicates that *xa_forget*( ) shall be performed asynchronously. Upon success, the function returns a positive value (called a handle) that the caller can use as an argument to *xa_complete*( ) to wait for the operation to complete. If the calling thread of control already has an asynchronous operation pending at the same resource manager for the same XID, this function fails, returning [XAER_ASYNC].

TMNOFLAGS
: This flag must be used when no other flags are set in *flags*.

RETURN VALUE

The function *xa_forget*( ) has the following return values:

[XA_OK]
: Normal execution.

[XAER_ASYNC]
: TMASYNC was set in *flags*, and either the maximum number of outstanding asynchronous operations has been exceeded, or TMUSEASYNC is not set in the *flags* element of the resource manager's **xa_switch_t** structure.

[XAER_RMERR]
: An error occurred in the resource manager and the resource manager has not forgotten the transaction branch.

[XAER_RMFAIL]
: An error occurred that makes the resource manager unavailable.

[XAER_NOTA]
: The specified XID is not known by the resource manager as a heuristically completed XID.

[XAER_INVAL]
: Invalid arguments were specified.

[XAER_PROTO]
: The routine was invoked in an improper context. See Chapter 6 for details.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The function returns a positive value upon success if the caller set TMASYNC (see above) in *flags*.

SEE ALSO

xa_commit( ), *xa_complete*( ), *xa_open*( ), *xa_recover*( ), *xa_rollback*( ).

WARNINGS

From the resource manager's perspective, the pointer *xid* is valid only for the duration of the call to *xa_forget*( ). That is, once the function completes, either synchronously or asynchronously, the transaction manager is allowed to invalidate where *xid* points. Resource managers are encouraged to use private copies of **xid* after the function completes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

xa_open — open a resource manager

SYNOPSIS

```
#include "xa.h"

int
xa_open(char *xa_info, int rmid, long flags)
```

DESCRIPTION

A transaction manager calls *xa_open*( ) to initialise a resource manager and prepare it for use in a distributed transaction processing environment. It applies to resource managers that support the notion of *open* and must be called before any other resource manager (*xa_*'()) calls are made.

The argument *xa_info* points to a null-terminated character string that may contain instance-specific information for the resource manager. The maximum length of this string is 256 bytes (including the null terminator). The argument *xa_info* may point to an empty string if the resource manager does not require instance-specific information to be available. The argument *xa_info* must not be a null pointer.

The argument *rmid*, an integer assigned by the transaction manager, uniquely identifies the called resource manager instance within the thread of control. The transaction manager passes the *rmid* on subsequent calls to XA routines to identify the resource manager. This identifier remains constant until the transaction manager in this thread closes the resource manager.

If the resource manager supports multiple instances, the transaction manager can call *xa_open*( ) more than once for the same resource manager. The transaction manager generates a new *rmid* value for each call, and must use different values for **xa_info* on each call, typically to identify the respective resource domain.

A transaction manager must call this function from the same thread of control that accesses the resource manager. In addition, attempts to open a resource manager instance that is already open have no effect and return success, [XA_OK].

The function's last argument, *flags*, must be set to one of the following values:

TMASYNC
: This flag indicates that *xa_open*( ) shall be performed asynchronously. Upon success, the function returns a positive value (called a handle) that the caller can use as an argument to *xa_complete*( ) to wait for the operation to complete. If the calling thread of control already has an asynchronous operation pending at the same resource manager, this function fails, returning [XAER_ASYNC].

TMNOFLAGS
: This flag must be used when no other flags are set in *flags*.

RETURN VALUE

The function *xa_open*( ) has the following return values:

[XA_OK]
: Normal execution.

[XAER_ASYNC]
: TMASYNC was set in *flags*, and either the maximum number of outstanding asynchronous operations has been exceeded, or TMUSEASYNC is not set in the *flags* element of the resource manager's **xa_switch_t** structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[XAER_RMERR]
An error occurred when opening the resource.

[XAER_INVAL]
Invalid arguments were specified.

[XAER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

The function returns a positive value upon success if the caller set TMASYNC (see above) in *flags*.

SEE ALSO

xa_close( ), *xa_complete*( ).

WARNINGS

From the resource manager's perspective, the pointer *xa_info* is valid only for the duration of the call to *xa_open*( ). That is, once the function completes, either synchronously or asynchronously, the transaction manager is allowed to invalidate where *xa_info* points. Resource managers are encouraged to use private copies of **xa_info* after the function completes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

xa_prepare — prepare to commit work done on behalf of a transaction branch

SYNOPSIS

```
#include "xa.h"

int
xa_prepare(XID *xid, int rmid, long flags)
```

DESCRIPTION

A transaction manager calls *xa_prepare*( ) to request a resource manager to prepare for commitment any work performed on behalf of the transaction identified by **xid*. The resource manager places any resources it held or modified in such a state that it can make the results permanent when it receives a commit request (that is, when the transaction manager calls *xa_commit*( )). If the transaction branch has already been prepared with *xa_prepare*( ), subsequent calls to *xa_prepare*( ) return [XAER_PROTO]. A transaction manager may call this function from any thread of control. All associations for the transaction identified by **xid* must have been ended by using *xa_end*( ) with TMSUCCESS set in *flags*.

If the transaction manager receives a successful completion, the resource manager guarantees that the transaction branch can be either committed or rolled back, regardless of failures. A resource manager shall not lose the branch information needed to perform a commit or rollback until the transaction manager calls either *xa_commit*( ) or *xa_rollback*( ) to complete the branch.

As an optimisation, a resource manager may indicate either that the work performed on behalf of a transaction branch was read-only or that the resource manager was not accessed on behalf of a branch (that is, *xa_prepare*( ) may return [XA_RDONLY]). In either case, the resource manager may release all resources and forget about the branch.

A transaction manager must issue a separate *xa_prepare*( ) for each transaction branch that accessed the resource manager on behalf of the global transaction.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

The function's last argument, *flags*, must be set to one of the following values:

TMASYNC
: This flag indicates that *xa_prepare*( ) shall be performed asynchronously. Upon success, the function returns a positive value (called a handle) that the caller can use as an argument to *xa_complete*( ) to wait for the operation to complete. If the calling thread of control already has an asynchronous operation pending at the same resource manager for the same XID, this function fails, returning [XAER_ASYNC].

TMNOFLAGS
: This flag must be used when no other flags are set in *flags*.

RETURN VALUE

The function *xa_prepare*( ) has the following return values:

[XA_RDONLY]
: The transaction branch was read-only and has been committed.

[XA_OK]
: Normal execution.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[XA_RB*]
The resource manager did not prepare to commit the work done on behalf of the transaction branch. Upon return, the resource manager has rolled back the branch's work and has released all held resources. The following values may be returned:

[XA_RBROLLBACK]
The resource manager rolled back the transaction branch for an unspecified reason.

[XA_RBCOMMFAIL]
A communication failure occurred within the resource manager.

[XA_RBDEADLOCK]
The resource manager detected a deadlock.

[XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.

[XA_RBOTHER]
The resource manager rolled back the transaction branch for a reason not on this list.

[XA_RBPROTO]
A protocol error occurred within the resource manager.

[XA_RBTIMEOUT]
The work represented by this transaction branch took too long.

[XA_RBTRANSIENT]
The resource manager detected a transient error.

[XAER_ASYNC]
TMASYNC was set in *flags*, and either the maximum number of outstanding asynchronous operations has been exceeded, or TMUSEASYNC is not set in the *flags* element of the resource manager's **xa_switch_t** structure.

[XAER_RMERR]
The resource manager encountered an error in preparing to commit the transaction branch's work. The specified XID may or may not have been prepared.

[XAER_RMFAIL]
An error occurred that makes the resource manager unavailable. The specified XID may or may not have been prepared.

[XAER_NOTA]
The specified XID is not known by the resource manager.

[XAER_INVAL]
Invalid arguments were specified.

[XAER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

The function returns a positive value upon success if the caller set TMASYNC (see above) in *flags*.

SEE ALSO

xa_commit( ), *xa_complete*( ), *xa_open*( ), *xa_rollback*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

WARNINGS

From the resource manager's perspective, the pointer *xid* is valid only for the duration of the call to *xa_prepare*( ). That is, once the function completes, either synchronously or asynchronously, the transaction manager is allowed to invalidate where *xid* points. Resource managers are encouraged to use private copies of **xid* after the function completes.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

xa_recover — obtain a list of prepared or heuristically completed transaction branches from a resource manager

SYNOPSIS

```
#include "xa.h"

int
xa_recover(XID *xids, long count, int rmid, long flags)
```

DESCRIPTION

A transaction manager calls *xa_recover*( ) during recovery to obtain a list of transaction branches that are currently in a prepared or heuristically completed state. The caller points *xids* to an array into which the resource manager places XIDs for these transactions, and sets *count* to the maximum number of XIDs that fit into that array.

So that all XIDs may be returned irrespective of the size of the array *xids*, one or more *xa_recover*( ) calls may be used within a single recovery scan. The *flags* parameter to *xa_recover*( ) defines when a recovery scan should start or end, or start and end. The start of a recovery scan moves a cursor to the start of a list of prepared and heuristically completed transactions. Throughout the recovery scan the cursor marks the current position in that list. Each call advances the cursor past the set of XIDs it returns.

Two consecutive complete recovery scans return the same list of transaction branches unless a transaction manager calls *xa_commit*( ), *xa_forget*( ), *xa_prepare*( ), or *xa_rollback*( ) for that resource manager, or unless that resource manager heuristically completes some branches, between the two recovery scans.

A transaction manager may call this function from any thread of control, but all calls in a given recovery scan must be made by the same thread.

Upon success, *xa_recover*( ) places zero or more XIDs in the space pointed to by *xids*. The function returns the number of XIDs it has placed there. If this value is less than *count*, there are no more XIDs to recover and the current scan ends. (That is, the transaction manager need not call *xa_recover*( ) again with TMENDRSCAN set in *flags*.) Multiple invocations of *xa_recover*( ) retrieve all the prepared and heuristically completed transaction branches.

It is the transaction manager's responsibility to ignore XIDs that do not belong to it.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

Following are the valid settings of *flags*:

TMSTARTRSCAN
: This flag indicates that *xa_recover*( ) should start a recovery scan for the thread of control and position the cursor to the start of the list. XIDs are returned from that point. If a recovery scan is already open, the effect is as if the recovery scan were ended and then restarted.

TMENDRSCAN
: This flag indicates that *xa_recover*( ) should end the recovery scan after returning the XIDs. If this flag is used in conjunction with TMSTARTRSCAN, the single *xa_recover*( ) call starts and then ends a scan.

TMNOFLAGS
: This flag must be used when no other flags are set in *flags*. A recovery scan must already be started. XIDs are returned starting at the current cursor position.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RETURN VALUE

The function *xa_recover*( ) has the following return values:

[≥ 0]
As a return value, *xa_recover*( ) normally returns the total number of XIDs it returned in **xids*.

[XAER_RMERR]
An error occurred in determining the XIDs to return.

[XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.

[XAER_INVAL]
The pointer *xids* is NULL and *count* is greater than 0, *count* is negative, an invalid *flags* was specified, or the thread of control does not have a recovery scan open and did not specify TMSTARTRSCAN in *flags*.

[XAER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

SEE ALSO

xa_commit( ), *xa_forget*( ), *xa_open*( ), *xa_rollback*( ).

WARNINGS

If *xids* points to a buffer that cannot hold all of the XIDs requested, *xa_recover*( ) may overwrite the caller's data space.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

xa_rollback — roll back work done on behalf of a transaction branch

SYNOPSIS

```
#include "xa.h"

int
xa_rollback(XID *xid, int rmid, long flags)
```

DESCRIPTION

A transaction manager calls *xa_rollback*( ) to roll back the work performed at a resource manager on behalf of the transaction branch. A branch must be capable of being rolled back until it has successfully committed. Any resources held by the resource manager for the branch are released and those modified are restored to their values at the start of the branch. A transaction manager may call this function from any thread of control.

Once rolled back, a resource manager can forget about the transaction branch so long as all threads of control currently accessing the resource manager on behalf of the transaction identified by **xid* have been notified of the branch's failure. The transaction manager must ensure that no new threads of control are allowed to access a resource manager with a rolled back (or marked rollback-only) XID.

In addition, *xa_rollback*( ) must guarantee that resources can eventually be released and restored to their initial values. That is, because this function may be used by a transaction manager to resolve deadlocks (defined in a manner dependent on the transaction manager), *xa_rollback*( ) must not itself be susceptible to indefinite blocking.

If a resource manager already completed the work associated with **xid* heuristically, this function merely reports how the resource manager completed the transaction branch. A resource manager shall not forget about a heuristically completed branch until the transaction manager calls *xa_forget*( ). Otherwise, indeterminate behaviour will result.

A transaction manager must issue a separate *xa_rollback*( ) for each transaction branch that accessed the resource manager on behalf of the global transaction.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

The function's last argument, *flags*, must be set to one of the following values:

TMASYNC
: This flag indicates that *xa_rollback*( ) shall be performed asynchronously. Upon success, the function returns a positive value (called a handle) that the caller can use as an argument to *xa_complete*( ) to wait for the operation to complete. If the calling thread of control already has an asynchronous operation pending at the same resource manager for the same XID, this function fails, returning [XAER_ASYNC].

TMNOFLAGS
: This flag must be used when no other flags are set in *flags*.

RETURN VALUE

The function *xa_rollback*( ) has the following return values:

[XA_HEURHAZ]
: Due to some failure, the work done on behalf of the specified transaction branch may have been heuristically completed. A resource manager shall return this value only if it has successfully prepared the transaction identified by **xid*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[XA_HEURCOM]
Due to a heuristic decision, the work done on behalf of the specified transaction branch was committed. A resource manager shall return this value only if it has successfully prepared the transaction identified by **xid*.

[XA_HEURRB]
Due to a heuristic decision, the work done on behalf of the specified transaction branch was rolled back. A resource manager shall return this value only if it has successfully prepared the transaction identified by **xid*.

[XA_HEURMIX]
Due to a heuristic decision, the work done on behalf of the specified transaction branch was partially committed and partially rolled back. A resource manager shall return this value only if it has successfully prepared the transaction identified by **xid*.

[XA_OK]
Normal execution.

[XA_RB*]
The resource manager has rolled back the transaction branch's work and has released all held resources. These values are typically returned when the branch was already marked rollback-only. The following values may be returned:

[XA_RBROLLBACK]
The resource manager rolled back the transaction branch for an unspecified reason.

[XA_RBCOMMFAIL]
A communication failure occurred within the resource manager.

[XA_RBDEADLOCK]
The resource manager detected a deadlock.

[XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.

[XA_RBOTHER]
The resource manager rolled back the transaction branch for a reason not on this list.

[XA_RBPROTO]
A protocol error occurred within the resource manager.

[XA_RBTIMEOUT]
The work represented by this transaction branch took too long.

[XA_RBTRANSIENT]
The resource manager detected a transient error.

[XAER_ASYNC]
TMASYNC was set in *flags*, and either the maximum number of outstanding asynchronous operations has been exceeded, or TMUSEASYNC is not set in the *flags* element of the resource manager's **xa_switch_t** structure.

[XAER_RMERR]
An error occurred in rolling back the transaction branch. The resource manager is free to forget about the branch when returning this error so long as all accessing threads of control have been notified of the branch's state.

[XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[XAER_NOTA]
The specified XID is not known by the resource manager.

[XAER_INVAL]
Invalid arguments were specified.

[XAER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

The function returns a positive value upon success if the caller set TMASYNC (see above) in *flags*.

SEE ALSO

xa_commit( ), *xa_complete*( ), *xa_forget*( ), *xa_open*( ), *xa_prepare*( ).

WARNINGS

From the resource manager's perspective, the pointer *xid* is valid only for the duration of the call to *xa_rollback*( ). That is, once the function completes, either synchronously or asynchronously, the transaction manager is allowed to invalidate where *xid* points. Resource managers are encouraged to use private copies of **xid* after the function completes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

xa_start — start work on behalf of a transaction branch

SYNOPSIS

```
#include "xa.h"

int
xa_start(XID *xid, int rmid, long flags)
```

DESCRIPTION

A transaction manager calls *xa_start*( ) to inform a resource manager that an application may do work on behalf of a transaction branch. Since many threads of control can participate in a branch and each one may be invoked more than once, *xa_start*( ) must recognise whether or not the XID exists. If another thread is accessing the calling thread's resource manager for the same branch, *xa_start*( ) may block and wait for the active thread to release control of the branch (via *xa_end*( )). A transaction manager must call this function from the same thread of control that accesses the resource manager. If the resource manager is doing work outside any global transaction on behalf of the application, *xa_start*( ) returns [XAER_OUTSIDE].

A transaction manager calls *xa_start*( ) only for those resource managers that do not have TMREGISTER set in the *flags* element of their **xa_switch_t** structure. Resource managers with TMREGISTER set must use *ax_reg*( ) to join a transaction branch (see *ax_reg*( ) for details).

The first argument, *xid*, is a pointer to the XID that a resource manager must associate with the calling thread of control. The transaction manager guarantees the XID to be unique for different transaction branches. The transaction manager may generate a new branch qualifier within the XID when it calls *xa_start*( ) for a new thread of control association (that is, when TMRESUME is not set in *flags*; see TMRESUME below). If the transaction manager elects to reuse a branch qualifier previously given to the resource manager for the XID, the transaction manager must inform the resource manager that it is doing so (by setting TMJOIN in *flags*; see TMJOIN below).

If the transaction manager generates a new branch qualifier in the XID, this thread is loosely-coupled to the other threads in the same branch. That is, the resource manager may treat this thread's work as a separate global transaction with respect to its isolation policies. If the transaction manager reuses a branch qualifier in the XID, this thread is tightly-coupled to the other threads that share the branch. An RM must guarantee that tightly-coupled threads are treated as a single entity with respect to its isolation policies and that no deadlock occurs within the branch among these tightly-coupled threads. Otherwise, indeterminate behaviour will result.

The argument *rmid*, the same integer that the transaction manager generated when calling *xa_open*( ), identifies the resource manager called from the thread of control.

Following are the valid settings of *flags*:

TMJOIN

This flag indicates that the thread of control is joining the work of an existing transaction branch. The resource manager shall make available enough transaction context so that tightly-coupled threads are not susceptible to resource deadlock within the branch.

If a resource manager does not recognise the transaction identified by **xid*, the function fails, returning [XAER_NOTA]. Note that this flag cannot be used in conjunction with TMRESUME.

TMRESUME

This flag indicates that a thread of control is resuming work on the specified transaction branch. The resource manager shall make available at least the transaction context that is specific to the resource manager, present at the time of the suspend, as if the thread had

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

effectively never been suspended, except that other threads in the global transaction may have affected this context.

If a resource manager does not recognise the transaction identified by **xid*, the function fails, returning [XAER_NOTA]. If the resource manager allows an association to be resumed in a different thread from the one that suspended the work, and the transaction manager expressed its intention to migrate the association (via the TMMIGRATE flag on *xa_end*( )), the current thread may be different from the one that suspended the work. Otherwise, the current thread must be the same, or the resource manager returns [XAER_PROTO]. When TMRESUME is set, the transaction manager uses the same XID it used in the *xa_end*( ) call that suspended the association.

If **xid* contains a reused branch qualifier, and the transaction manager has multiple outstanding suspended thread associations for **xid*, the following rules apply:

- The transaction manager can have only one of them outstanding at any time with TMMIGRATE set in *flags*.
- Moreover, the transaction manager cannot resume this association in a thread that currently has a non-migratable suspended association.

These rules prevent ambiguity as to which context is restored.

TMNOWAIT
When this flag is set and a blocking condition exists, *xa_start*( ) returns [XA_RETRY] and the resource manager does not associate the calling thread of control with **xid* (that is, the call has no effect). Note that this flag cannot be used in conjunction with TMASYNC.

TMASYNC
This flag indicates that *xa_start*( ) shall be performed asynchronously. Upon success, the function returns a positive value (called a handle) that the caller can use as an argument to *xa_complete*( ) to wait for the operation to complete. If the calling thread of control already has an asynchronous operation pending at the same resource manager, this function fails, returning [XAER_ASYNC].

TMNOFLAGS
This flag must be used when no other flags are set in *flags*.

RETURN VALUE

The function *xa_start*( ) has the following return values:

[XA_RETRY]
TMNOWAIT was set in *flags* and a blocking condition exists.

[XA_OK]
Normal execution.

[XA_RB*]
The resource manager has not associated the transaction branch with the thread of control and has marked the transaction identified by **xid* as rollback-only. The following values may be returned regardless of the setting of *flags*:

[XA_RBROLLBACK]
The resource manager marked the transaction branch rollback-only for an unspecified reason.

[XA_RBCOMMFAIL]
A communication failure occurred within the resource manager.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[XA_RBDEADLOCK]
The resource manager detected a deadlock.

[XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.

[XA_RBOTHER]
The resource manager marked the transaction branch rollback-only for a reason not on this list.

[XA_RBPROTO]
A protocol error occurred within the resource manager.

[XA_RBTIMEOUT]
The work represented by this transaction branch took too long.

[XA_RBTRANSIENT]
The resource manager detected a transient error.

[XAER_ASYNC]
TMASYNC was set in *flags*, and either the maximum number of outstanding asynchronous operations has been exceeded, or TMUSEASYNC is not set in the *flags* element of the resource manager's **xa_switch_t** structure.

[XAER_RMERR]
An error occurred in associating the transaction branch with the thread of control.

[XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.

[XAER_DUPID]
If neither TMRESUME nor TMJOIN was set in *flags* (indicating the initial use of **xid*) and the XID already exists within the resource manager, the resource manager must return [XAER_DUPID]. The resource manager failed to associate the transaction branch with the thread of control.

[XAER_OUTSIDE]
The resource manager is doing work outside any global transaction on behalf of the application.

[XAER_NOTA]
Either TMRESUME or TMJOIN was set in *flags*, and the specified XID is not known by the resource manager.

[XAER_INVAL]
Invalid arguments were specified.

[XAER_PROTO]
The routine was invoked in an improper context. See Chapter 6 for details.

The function returns a positive value upon success if the caller set TMASYNC (see above) in *flags*.

SEE ALSO

ax_reg( ), *xa_complete*( ), *xa_end*( ), *xa_open*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

WARNINGS

From the resource manager's perspective, the pointer *xid* is valid only for the duration of the call to *xa_start*( ). That is, once the function completes, either synchronously or asynchronously, the transaction manager is allowed to invalidate where *xid* points. Resource managers are encouraged to use private copies of **xid* after the function completes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Chapter 6: State Tables

This chapter contains state tables that show legal calling sequences for the XA routines. TMs must sequence their use of the XA routines so that the calling thread of control makes legal transitions through each applicable table in this chapter. That is, any TM must, on behalf of each AP thread of control:

- open and close each RM as shown in Table 6-1 on page 58
- associate itself with, and dissociate itself from, transaction branches as shown in Table 6-2 on page 59 or Table 6-3 on page 60, whichever applies
- advance any transaction branch toward completion through legal transitions as shown in Table 6-4 on page 62
- make legal transitions through Table 6-5 on page 63, whenever the TM calls XA routines in the asynchronous mode.

Table 6-5 on page 63 is the only table that addresses the asynchronous mode of XA routines. The other tables also describe routines that can be called asynchronously. In this case, the tables view the XA call that initiates an operation, and the *xa_complete*( ) call that shows that the operation is complete, as a single event.

Interpretation of the Tables

A single call may make transitions in more than one of the state tables. Services that are not pertinent to a given state table are omitted from that table.

All the tables describe the state of a thread of control with respect to a particular RM. That is, the tables indicate the validity of a sequence of XA calls only if the calls all pertain to the same RM. The thread of control could be dealing with other RMs at the same time, which might be in entirely different states. Each state table indicates the valid initial state or states for such a sequence; it is not always the leftmost state (the state with the zero subscript).

Table 6-2 on page 59, Table 6-3 on page 60, and Table 6-4 on page 62 describe the sequence of calls with respect to the progress of a particular XID. Other XIDs within the same RM thread of control may be in different states as they progress from initial creation through completion, except that a thread can have only one active association at a time. Thus, while one XID may be actively associated in a thread of control, the same thread of control may make branch completion calls for other XIDs.

An entry under a particular state in the table asserts that an XA routine can be called in that state, and shows the resulting state. A blank entry asserts that it is an error to call the routine in that state. The routine should return the protocol error [XAER_PROTO], unless another error code that gives more specific information also applies.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Notation

Sometimes a routine makes a state transition only when the caller gives it certain input, or only when the routine returns certain output (such as a return code). Specific state table entries describe these cases. The tables describe input to the routine in parentheses, even though that may not be the exact syntax used; for example, *xa_end*(TMFAIL) describes a call to *xa_end*( ) in which the caller sets the TMFAIL bit in the *flags* argument. The tables denote output from the routine, including return status, using an arrow (→) followed by the specific output.

For example, the legend

xa_end () → [XA_RB]

describes the case where a call to *xa_end*( ) returns one of the [XA_RB*] codes.

A general state table entry (one that does not show flags or output values) describes all remaining cases of calls to that routine. These general entries assume the routine returns success. (The *xa_**( ) routines return the [XA_OK] code; the *ax_**( ) routines return [TM_OK].) Calls that return failure do not make state transitions, except where described by specific state table entries.

The notation *xa_**( ) refers to all applicable *xa_* routines.

6.1 Resource Manager Initialisation

For each thread of control, each RM is either open or closed. The *initial state* is closed (R_0). The *xa_open*( ) and *xa_close*( ) routines move an RM between these states. Redundant uses of these routines are valid, as Table 6-1 shows:

Table 6-1 State Table for Resource Manager Initialisation

XA Routines	Resource Manager States	
	Un-initialised R_0	**Initialised** R_1
xa_open( )	R_1	R_1
xa_close( )	R_0	R_0

A transition to R_1 enables the use of Table 6-2 on page 59 to Table 6-4 on page 62 inclusive. The state R_0 appears in these tables to illustrate that closing an RM precludes its use in global transactions. At this point, Table 6-1 governs legal sequences.

In Table 6-2 on page 59 to Table 6-4 on page 62 a return of [XAER_RMFAIL] on any routine causes a state transition in that thread to state R_0.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.2 Association of Threads of Control with Transactions

Table 6-2 shows the state of an association between a thread of control and a transaction branch. (See Table 6-3 on page 60 for RMs that dynamically register with a TM.) Valid initial states of association for a thread of control are T_0 and T_2. (The *Association Suspended* state, T_2, includes the case where a thread has suspended an association migratably. After returning to T_0, any thread can re-enter this table in column T_2 to resume or end that other association.)

Table 6-2 makes the following assumptions:

- The calling thread remains in state R_1, except where R_0 is explicitly indicated.
- The RM does not have TMREGISTER set in its switch.
- The caller passes the same *rmid* and XID as arguments to each applicable routine listed below.

Table 6-2 shows the effect of *xa_start*( ) and *xa_end*( ) on the thread of control's association with a single transaction branch. These routines may also change the state of the branch itself. Therefore, Table 6-4 on page 62 further constrains their use.

If a thread suspends its association, it can perform work on behalf of other transaction branches before resuming the suspended association.

Table 6-2 State Table for Transaction Branch Association

XA Routines	Transaction Branch Association States		
	Not Associated T_0	**Associated** T_1	**Association Suspended** T_2
xa_start( )	T_1		
xa_start(TMRESUME)			T_1
xa_start(TMRESUME) → [XA_RB]			T_0
xa_end(TMSUSPEND)		T_2	
xa_end(TMSUSPEND) → [XA_RB]		T_0	
xa_end(TMSUCCESS)		T_0	T_0
xa_end(TMFAIL)		T_0	T_0
xa_open( )	T_0	T_1	T_2
xa_recover( )	T_0	T_1	T_2
xa_close( )	R_0		R_0
*xa_**( ) → [XAER_RMFAIL]	R_0	R_0	R_0

6.2.1 Dynamic Registration of Threads

Table 6-3 on page 60 shows the state of an association between a thread of control and a transaction branch. This table is for RMs that dynamically register with a TM. Valid initial states in Table 6-3 on page 60 are D_0 and D_2. (The Association Suspended state, D_2, includes the case where a thread has suspended an association migratably. After returning to D_0, any thread can re-enter this table in column D_2 to resume or end that other association.)

The top half of the table shows the legal sequence of calls for an RM thread of control. The bottom half of the table shows the legal sequence of calls for a TM thread of control. The thread of control calling these routines must comply with the applicable half of the table.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Table 6-3 makes the following assumptions:

- The TM calling thread remains in state R_1, except where R_0 is explicitly indicated.
- The RM has TMREGISTER set in its switch.
- The caller passes the same *rmid* and XID as arguments to each applicable routine listed below.
- The same RM and XID are used for the dynamic registration functions.

Table 6-3 defines the behaviour of a single transaction branch in a thread. If a thread suspends its association, it can perform work on behalf of other branches before resuming the suspended association.

Table 6-3 State Table for Transaction Branch Association (Dynamic Registration)

XA Routines	Transaction Branch Association States			
	Not Registered D_0	**Registered with Valid XID** D_1	**Registration Suspended** D_2	**Registered with NULLXID** D_3
Resource Manager Calls				
ax_reg( ) → valid XID	D_1			
ax_reg( ) → NULLXID	D_3			
ax_reg( ) → [TM_RESUME]			D_1	
ax_unreg( )				D_0
Transaction Manager Calls				
xa_end(TMSUSPEND)		D_2		
xa_end(TMSUSPEND) → [XA_RB]		D_0		
xa_end(TMSUCCESS)		D_0	D_0	
xa_end(TMFAIL)		D_0	D_0	
xa_open( )	D_0	D_1	D_2	D_3
xa_recover( )	D_0	D_1	D_2	D_3
xa_close( )	R_0		R_0	
*xa_**( ) → [XAER_RMFAIL]	R_0	R_0	R_0	R_0

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.3 Transaction States

Table 6-4 on page 62 shows the commitment protocol for a transaction branch. Any state listed in Table 6-4 on page 62, except state S_1 is a valid initial state for a thread of control. The table applies to sequential calls by a thread of control that:

- remains in state R_1, except where R_0 is explicitly indicated
- passes the same *XID* in each *xa_* call that requires an *XID.*

Table 6-4 on page 62 shows the effect of *xa_start*( ) and *xa_end*( ) on the state of a transaction branch. These routines may also change the thread's association with the branch. Therefore, Table 6-2 on page 59 and Table 6-3 on page 60 further constrain their use.

Table 6-4 on page 62 does not apply to uses of *xa_end*(TMSUSPEND), *xa_start*(TMRESUME), or *ax_reg* in which [TM_RESUME] is returned; the uses of these are constrained by Table 6-2 on page 59 and Table 6-3 on page 60 and the following rules:

- *xa_end*(TMSUSPEND) and *xa_end*(TMMIGRATE) may only be used if no other thread association for this branch was suspended with the TMMIGRATE flag. (This rule ensures that there exists at most one migratable suspended association for a branch.)
- *xa_start*(TMRESUME) may be used, and *ax_reg* may return [TM_RESUME], only on a branch that has at least one suspended association. That suspended association must either have been suspended non-migratably by the acting thread or suspended migratably by any thread. If both conditions are true, the association which was suspended non-migratably by the acting thread is the one resumed.
- *xa_end*(TMSUCCESS) may be used only on a branch that is associated with the current thread or that has at least one suspended association. If the branch is associated with the current thread, it is that association which is ended. Otherwise, a suspended association ends, as though an implicit *xa_start*(TMRESUME) were performed (see above) before the *xa_end*(TMSUCCESS).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Table 6-4 State Table for Transaction Branches

	XA Routines	Transaction Branch States					
		Non-existent Transaction S_0	Active S_1	Idle S_2	Prepared S_3	Rollback Only S_4	Heuristically Completed S_5
†	*xa_start*( )	S_1		S_1			
‡	*xa_start*( ) → [XA_RB]			S_4			
	xa_end( )		S_2				
‡	*xa_end*( ) → [XA_RB]		S_4				
	xa_prepare( )			S_3			
	xa_prepare( ) →			S_0			
‡	[XA_RDONLY] or [XA_RB]						
	xa_prepare( ) → [XAER_RMERR]			S_2			
	xa_commit( ) → [XA_OK] or [XAER_RMERR]			S_0	S_0		
‡	*xa_commit*( ) → [XA_RB]			S_0			
	xa_commit( ) → [XA_RETRY]				S_3		
	xa_commit( ) →			S_5	S_5		S_5
‡	[XA_HEUR]						
	xa_rollback( ) →			S_0	S_0	S_0	
‡	[XA_OK] or [XA_RB] or [XAER_RMERR]						
	xa_rollback( ) →				S_5	S_5	S_5
‡	[XA_HEUR]						
	xa_forget( )						S_0
	xa_forget( ) → [XAER_RMERR]						S_5
	xa_open( )	S_0	S_1	S_2	S_3	S_4	S_5
	xa_recover( )	S_0	S_1	S_2	S_3	S_4	S_5
	xa_close( )	R_0		R_0	R_0	R_0	R_0
	xa_*( ) → [XAER_RMFAIL]	R_0	R_0	R_0	R_0	R_0	R_0

Notes:

† This row also applies when an applicable RM calls *ax_reg* and the TM informs the RM that its work is on behalf of a transaction branch.

‡ [XA_HEUR] denotes any of [XA_HEURCOM], [XA_HEURRB], [XA_HEURMIX], or [XA_HEURHAZ]. [XA_RB] denotes any return value with a prefix [XA_RB.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4 Asynchronous Operations

Table 6-5 describes asynchronous operations. The preceding tables do not take into account asynchronous operations.

Table 6-5 illustrates that once a TM thread makes an asynchronous request to an RM on behalf of an XID, the only valid request the TM thread can give the same RM for that XID is the corresponding *xa_complete*( ). For those routines that do not take an XID, the thread must wait for the operation's completion before issuing another request to that same RM. The only routines by which the TM can give additional work to the same RM are *xa_commit*( ), *xa_forget*( ), *xa_prepare*( ), and *xa_rollback*( ).

The valid **initial state** for a thread of control is A_0.

The asynchronous calls (with TMASYNC) achieve a state transition in Table 6-5 when the function returns a valid handle (a positive value). The table entries describing *xa_complete*( ) assume that the caller passes this same, valid handle to *xa_complete*( ).

Table 6-5 State Table for Asynchronous Operations

XA Routines	Asynchronous Operation States	
	Initial Call A_0	Operation Pending A_1
Any synchronous *xa_*( ) call	A_0	
xa_rollback(TMASYNC)	A_1	
xa_close(TMASYNC)	A_1	
xa_commit(TMASYNC)	A_1	
xa_end(TMASYNC)	A_1	
xa_forget(TMASYNC)	A_1	
xa_open(TMASYNC)	A_1	
xa_prepare(TMASYNC)	A_1	
xa_start(TMASYNC)	A_1	
xa_complete( )		A_0
xa_complete(TMNOWAIT)		A_0
xa_complete(TMNOWAIT) → [XA_RETRY]		A_1

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Chapter 7: Implementation Requirements

This chapter summarises the implications on implementors of this International Standard. It also identifies features of this International Standard that implementors of RMs or TMs can regard as optional.

These requirements are designed to facilitate portability — specifically, the ability to move a software component to a different DTP system without modifying the source code. It is anticipated that DTP products will be delivered as object modules and that the administrator will control the mix and operation of components at a particular site by:

1. re-linking object modules
2. supplying text strings to the software components (or executing a vendor-supplied procedure that generates suitable text strings).

With regard to re-linking, a conformant implementation may provide its routines in the form of object code or as skeleton routines which themselves invoke the appropriate routines in executable-image form.

7.1 Application Program Requirements

Any AP in a DTP system must use a TM and delegate to it responsibility to control and coordinate each global transaction. The X/Open **TX (Transaction Demarcation)** specification (published separately) provides the AP with the means of interfacing with a TM for this purpose.

The AP is not involved in either the commitment protocol or the recovery process. An AP thread can have only one global transaction active at a time.

The AP may ask for work to be done by calling one or more RMs. It uses the RM's native interface exactly as it would if no TM existed, except that it calls the TM to define global transactions (see Section 7.2.1 on page 68).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.2 Resource Manager Requirements

The X/Open DTP model affects only RMs operating in the DTP environment. The model puts these constraints on the architecture or implementation of the RM:

- **Interfaces**
  RMs must provide all the *xa_*()* routines specified in Chapter 3 for use by TMs, even if a particular routine requires no real action by that RM. RMs must also provide a native application program interface (see Section 7.2.1 on page 68). As this is the initial version of the XA interface, RMs must also set to 0 the version element of their switches.

  An RM in an executable module is linked to at most one TM.

- **Ability to recognise XIDs**
  RMs must accept XIDs from TMs. They must associate with the XID all the work they do for an AP. For example, if an RM identifies a thread of control using a process identifier, the RM must map the process identifier to the XID.

  An important attribute of the XID is global uniqueness, based on the exact order of the bits in the data portion of the XID for the lengths specified. Therefore, RMs must not alter in any way the bits in the data portion of the XID. For example, if an RM remotely communicates an XID, it must ensure that the data bits of the XID are not altered by the communication process. That is, the data part of the XID should be treated as an OCTET STRING (opaque data) using terminology defined in the referenced ASN.1 standard and the referenced BER standard.

- **Calling protocol**
  A single instance of an RM must allow multiple threads logically to gain access to it on behalf of the same transaction branch, although it has the option of actually implementing single-threaded access (for example, blocking a thread at the call to *xa_start*() or channelling all accesses through a single back-end process). An RM can use whatever it wishes from the AP's environment (for example, process ID, task ID, or user ID) to identify the calling thread.

  An RM must ensure that each calling thread makes *xa_*()* calls in a legal sequence, and must return [XAER_PROTO] or other suitable error if the caller violates the state tables (see Chapter 6).

- **Commitment protocol**
  The RM must support the commitment protocol specified in Section 2.3 on page 8. This has the following implications:

  — An RM must provide an *xa_prepare*() routine, and must be able to report whether it can guarantee its ability to commit the transaction branch. If it reports that it can, it must reserve all the resources needed to commit the branch. It must hold those resources until the TM directs it either to commit or roll back the branch.

  — An RM must support the one-phase commit optimisation (see Section 2.3.2 on page 8). That is, it must allow *xa_commit*() (specifying TMONEPHASE) even if it has not yet received *xa_prepare*() for the transaction branch in question.

- **Support for Recovery**
  An RM must track the status of all transaction branches in which it is involved. After responding affirmatively to the TM's *xa_prepare*() call, the RM cannot erase its knowledge of the branch, or of the work it has done in support of that branch, until it receives and successfully performs the TM's order to commit or roll back the branch. If an RM heuristically completes a branch, it cannot forget the branch until explicitly permitted to by the TM. On command, an RM must return a list of all its branches that it has prepared to commit or has heuristically completed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When an RM recovers from its own failure, it recovers prepared and heuristically completed transaction branches. It forgets all other branches.

- **Public information**
  An RM product must publish the following information:
  - The name of its **xa_switch_t** structure. This switch gives RM entry points and other information; Section 4.3 on page 21 specifies its structure. Multiple RMs may share the same **xa_switch_t** structure. Each pointer must point to an actual routine, even pointers that are theoretically unused on that RM. These routines must return in an orderly way, in case they are mistakenly called. (The RM does not need to publish the names of the individual routines.)
  - The text of the string, within the RM switch, that specifies the name of the RM.
  - The forms of the information strings that its *xa_open*( ) and *xa_close*( ) routines accept, and the way that different *xa_open*( ) strings specify different resource domains for different RM instances.
  - The names of libraries or object files, in the correct sequence, that the administrator must use when linking APs with the RM.
  - Any semantics in the native interface that affect global transaction processing — for example, specifying isolation level, transaction completion, or effects that differ from non-DTP operation.

  ISO will specify how an RM provider can guarantee that its names do not conflict with the name of any other RM product or version produced by that or another organisation.
- **Implementor options**
  Each RM has the option of implementing these features:
  - **Open/close informational strings**
    Any RM may accept a null string in place of the informational string argument on calls to its *xa_open*( ) and *xa_close*( ) routines. (If it does so, it must publish this fact.)
  - **Protocol optimisations**
    The read-only optimisation discussed in Section 2.3.2 on page 8 is optional.
  - **Association migration**
    An RM can allow a TM to resume a suspended association in a thread of control other than the one where the suspension occurred.
  - **Branch identification**
    An RM can use the *bqual* component of the XID structure to let different branches of the same global transaction prepare to commit at different times, and to avoid deadlock (see Section 4.2 on page 19).
  - **Dynamic registration**
    This feature, as described in Section 3.3.1 on page 15, is optional.
  - **Asynchrony**
    Support for the asynchronous mode discussed in Section 3.5 on page 17 is optional. If the TMUSEASYNC flag is set in the RM's switch, the RM must support the asynchronous mode. It may still complete some requests synchronously, either when the TM makes the original asynchronous call or when the TM calls *xa_complete*( ). If the TMUSEASYNC flag is not set in the RM's switch and the TM makes an asynchronous call, the RM must return [XAER_ASYNC].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— **Heuristics**
As described in the *xa_commit*( ) and *xa_rollback*( ) reference manual pages, an RM can report that it has heuristically completed the transaction branch. This feature is optional.

7.2.1 The Application Program (Native) Interface

RMs must provide a well-defined native interface that APs can use to request work. To maximise portability, all AP-RM interfaces should adhere to any applicable publication. For example, a relational DBMS RM should use the SQL defined in the ISO SQL standard (ISO/IEC 9075:1992).

In the DTP environment, RMs must rely on the TM to manage global transactions. Some RMs, such as some Indexed Sequential Access Method (ISAM) file managers, have no concept of transactions. So this is a new requirement, but it does not change the native interface. In other RMs, such as SQL RDBMSs, the native interface defines transactions. An AP must not use these services in a DTP context, since TMs have no knowledge of an RM's transaction. For example, the application program interface for an SQL RM in the DTP environment must not let use of these statements affect the global transaction:

```
EXEC SQL COMMIT WORK
EXEC SQL ROLLBACK WORK
```

In addition, any service in the native AP-RM interface that affects its own commitment, or any non-standard service that has an effect on transactions, must not be used within a global transaction.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.3 Transaction Manager Requirements

- **Service interfaces**
  TMs must use the *xa_*()* routines the RM provides (see Chapter 3) to coordinate the work of all the local RMs that the AP uses. TMs must call *xa_open*() and *xa_close*() on any local RM associated with the TM. Each TM must ensure that each of its *xa_*()* calls addresses the correct RM.

  TMs must be written to handle consistently any information or status that an RM can legally return. A TM must assume that it may be linked with RMs that use any RM options that this International Standard allows, including multiple RMs in a single AP object program, dynamic registration of RMs, RMs that make heuristic decisions, and RMs that use the read-only protocol optimisation.

- **Transaction identifiers**
  A TM must generate XIDs conforming to the structure described in Section 4.2 on page 19. They must be globally unique and must adequately describe the transaction branch. To guarantee global uniqueness, the TM should use an ISO OBJECT IDENTIFIER (see the referenced ASN.1 standard and the referenced BER standard) that the TM knows is unique within the *gtrid* component of the XID. The TM should also use an ISO OBJECT IDENTIFIER within the *bqual* field, but the same OBJECT IDENTIFIER can be used for all XIDs that a given TM generates. The *bqual* fields identify the TM that generated them, and identify the transaction branch with which they are associated.

  Failure to use the ISO OBJECT IDENTIFIER format for XIDs could cause interoperability problems when multiple TMs are either involved in the same global transaction or affect shared resources.

- **Public information**
  A TM product must publish the following information:

  — linking directions for producing an application object program — these directions must describe how to link RM and TM libraries, and how to incorporate the switch from each RM

  — instructions for generating or locating appropriate informational strings that the TM uses when it calls *xa_open*() or *xa_close*() for each RM

- **Implementor options**
  The one-phase commit protocol optimisation (see Section 2.3.2 on page 8) and the asynchronous and non-blocking modes for calling RMs (see Section 3.5 on page 17) are optional.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Appendix A: Complete Text of "xa.h"

This appendix specifies the complete text of an "**xa.h**" file in both ANSI C (see the referenced C standard) and Common Usage C.

```
/*
 * Start of xa.h header
 *
 * Define a symbol to prevent multiple inclusions of this header file
 */
#ifndef XA_H
#define XA_H
/*
 * Transaction branch identification: XID and NULLXID:
 */
#define XIDDATASIZE     128     /* size in bytes */
#define MAXGTRIDSIZE    64      /* maximum size in bytes of gtrid */
#define MAXBQUALSIZE    64      /* maximum size in bytes of bqual */
struct xid_t {
  long formatID;                /* format identifier */
  long gtrid_length;            /* value from 1 through 64 */
  long bqual_length;            /* value from 1 through 64 */
  char data[XIDDATASIZE];
  };
typedef struct xid_t XID;
/*
 * A value of -1 in formatID means that the XID is null.
 */
/*
 * Declarations of routines by which RMs call TMs:
 */
#ifdef __STDC__
extern int ax_reg(int, XID *, long);
extern int ax_unreg(int, long);
#else /* ifndef __STDC__ */
extern int ax_reg();
extern int ax_unreg();
#endif /* ifndef __STDC__ */
/*
 * XA Switch Data Structure
 */
#define RMNAMESZ        32      /* length of resource manager name, */
                                /* including the null terminator */
define MAXINFOSIZE      256     /* maximum size in bytes of xa_info
                                /* strings, including the null terminator*/
struct xa_switch_t {
  char name[RMNAMESZ];          /* name of resource manager */
  long flags;                   /* options specific to the resource manager */
  long version;                 /* must be 0 */
#ifdef __STDC__
  int (*xa_open_entry)(char *, int, long);  /* xa_open function pointer*/
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
  int (*xa_close_entry)(char *, int, long); /* xa_close function pointer*/
  int (*xa_start_entry)(XID *, int, long);  /* xa_start function pointer*/
  int (*xa_end_entry)(XID *, int, long);    /* xa_end function pointer*/
  int (*xa_rollback_entry)(XID *, int, long);
                                            /* xa_rollback function pointer*/
  int (*xa_prepare_entry)(XID *, int, long);/* xa_prepare function pointer*/
  int (*xa_commit_entry)(XID *, int, long); /* xa_commit function pointer*/
  int (*xa_recover_entry)(XID *, long, int, long);
                                            /* xa_recover function pointer*/
  int (*xa_forget_entry)(XID *, int, long); /* xa_forget function pointer*/
  int (*xa_complete_entry)(int *, int *, int, long);
                                            /* xa_complete function pointer*/
#else /* ifndef __STDC__ */
  int (*xa_open_entry)();                   /* xa_open function pointer */
  int (*xa_close_entry)();                  /* xa_close function pointer */
  int (*xa_start_entry)();                  /* xa_start function pointer */
  int (*xa_end_entry)();                    /* xa_end function pointer */
  int (*xa_rollback_entry)();               /* xa_rollback function pointer*/
  int (*xa_prepare_entry)();                /* xa_prepare function pointer */
  int (*xa_commit_entry)();                 /* xa_commit function pointer */
  int (*xa_recover_entry)();                /* xa_recover function pointer */
  int (*xa_forget_entry)();                 /* xa_forget function pointer */
  int (*xa_complete_entry)();               /* xa_complete function pointer*/
#endif /* ifndef __STDC__ */
  };
/*
 * Flag definitions for the RM switch
 */
#define TMNOFLAGS    0x00000000L     /* no resource manager features
                                        selected */
#define TMREGISTER   0x00000001L     /* resource manager dynamically
                                        registers */
#define TMNOMIGRATE  0x00000002L     /* resource manager does not support
                                        association migration */
#define TMUSEASYNC   0x00000004L     /* resource manager supports
                                        asynchronous operations */
/*
 * Flag definitions for xa_ and ax_ routines
 */
/* use TMNOFLAGS, defined above, when not specifying other flags */
#define TMASYNC      0x80000000L     /* perform routine asynchronously */
#define TMONEPHASE   0x40000000L     /* caller is using one-phase commit
                                        optimisation */
#define TMFAIL       0x20000000L     /* dissociates caller and marks
                                        transaction branch rollback-only */
#define TMNOWAIT     0x10000000L     /* return if blocking condition exists */
#define TMRESUME     0x08000000L     /* caller is resuming association
                                        with suspended transaction branch */
#define TMSUCCESS    0x04000000L     /* dissociate caller from transaction
                                        branch*/
#define TMSUSPEND    0x02000000L     /* caller is suspending, not ending,
                                        association */
#define TMSTARTRSCAN 0x01000000L     /* start a recovery scan */
#define TMENDRSCAN   0x00800000L     /* end a recovery scan */
#define TMMULTIPLE   0x00400000L     /* wait for any asynchronous operation */
#define TMJOIN       0x00200000L     /* caller is joining existing transaction
                                        branch */
#define TMMIGRATE    0x00100000L     /* caller intends to perform migration */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/*
 * ax_() return codes (transaction manager reports to resource manager)
 */
#define TM_JOIN         2     /* caller is joining existing transaction
                                 branch */
#define TM_RESUME       1     /* caller is resuming association with
                                 suspended transaction branch */
#define TM_OK           0     /* normal execution */
#define TMER_TMERR      -1    /* an error occurred in the transaction
                                 manager */
#define TMER_INVAL      -2    /* invalid arguments were given */
#define TMER_PROTO      -3    /* routine invoked in an improper context */
/*
 * xa_() return codes (resource manager reports to transaction manager)
 */
#define XA_RBBASE       100             /* the inclusive lower bound of the
                                           rollback codes */
#define XA_RBROLLBACK   XA_RBBASE       /* the rollback was caused by an
                                           unspecified reason */
#define XA_RBCOMMFAIL   XA_RBBASE+1     /* the rollback was caused by a
                                           communication failure */
#define XA_RBDEADLOCK   XA_RBBASE+2     /* a deadlock was detected */
#define XA_RBINTEGRITY  XA_RBBASE+3     /* a condition that violates the
                                           integrity of the resources
                                           was detected */
#define XA_RBOTHER      XA_RBBASE+4     /* the resource manager rolled back
                                           the transaction branch for
                                           a reason not on this list */
#define XA_RBPROTO      XA_RBBASE+5     /* a protocol error occurred in the
                                           resource manager */
#define XA_RBTIMEOUT    XA_RBBASE+6     /* a transaction branch took too
                                           long */
#define XA_RBTRANSIENT  XA_RBBASE+7     /* may retry the transaction branch */
#define XA_RBEND        XA_RBTRANSIENT  /* the inclusive upper bound of the
                                           rollback codes */

#define XA_NOMIGRATE    9     /* resumption must occur where
                                 suspension occurred */
#define XA_HEURHAZ      8     /* the transaction branch may have
                                 been heuristically completed */
#define XA_HEURCOM      7     /* the transaction branch has been
                                 heuristically committed */
#define XA_HEURRB       6     /* the transaction branch has been
                                 heuristically rolled back */
#define XA_HEURMIX      5     /* the transaction branch has been
                                 heuristically committed and rolled back */
#define XA_RETRY        4     /* routine returned with no effect and
                                 may be reissued */
#define XA_RDONLY       3     /* the transaction branch was read-only and
                                 has been committed */
#define XA_OK           0     /* normal execution */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
#define XAER_ASYNC      -2   /* asynchronous operation already outstanding */
#define XAER_RMERR      -3   /* a resource manager error occurred in the
                             transaction branch */
#define XAER_NOTA       -4   /* the XID is not valid */
#define XAER_INVAL      -5   /* invalid arguments were given */
#define XAER_PROTO      -6   /* routine invoked in an improper context */
#define XAER_RMFAIL     -7   /* resource manager unavailable */
#define XAER_DUPID      -8   /* the XID already exists */
#define XAER_OUTSIDE    -9   /* resource manager doing work outside */
                             /* global transaction */

#endif /* ifndef XA_H */
/*
 * End of xa.h header
 */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Appendix B: DTP Model — Introduction

(For informative purposes, this appendix provides a copy of Chapter 1 of the published X/Open Guide: Distributed Transaction Processing Reference Model Version 3.)

B.1 Overview

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.

These terms are defined more precisely in Section D.2 on page 83.

X/Open DTP publications based on this model specify a portable high-level TP language (HTL), portable APIs and system-level interfaces that facilitate:

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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.2 Benefits of X/Open DTP

Distributed transaction processing provides the necessary mechanism to combine multiple software components into a cooperating unit that can maintain shared data, potentially spanning multiple physical processors or locations or both. This enables construction of applications that manipulate data consistently using multiple products, that can easily incorporate additional components, and that can be scaled by adding additional hardware and software components.

Portability, interchangeability, and interoperability let application writers benefit from a wider selection of transaction managers, resource managers, and communication resource managers. Application writers also gain the freedom to select from alternative products, hardware, and software platforms.

Published language and interface specifications simplify software design. For example, some software products that might once have been implemented using customised interfaces may now be viewed as interchangeable DTP components. This approach shortens development time and extends the above benefits to a greater number of products.

All parties benefit from the X/Open method of achieving consensus and communication among open systems providers, with a significant voice for system vendors, independent software vendors (ISVs), and users.

B.3 Areas Not Addressed

The X/Open DTP model does not address all issues of importance in transaction processing, for example:

- configuration, administration, and monitoring of transaction processing systems
- forms management and other user interfaces
- specification of benchmarks
- security
- naming
- communication techniques that may be used within RM implementations.

B.4 Relationship to International Standards

The X/Open DTP model shows that DTP software components use the communication protocol specified in the referenced OSI TP standards to facilitate interoperability of components in heterogeneous environments. The use of OSI TP is not mandatory where implementors require to use a product-internal communication provider.

Data structures from the referenced OSI CCR standards are a recommended component of the transaction identifier that X/Open specifies.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Appendix C: DTP Model — Definitions

(For informative purposes, this appendix provides a copy of Chapter 2 of the published X/Open Guide: Distributed Transaction Processing Reference Model Version 3.)

This chapter gives fundamental definitions on which subsequent chapters depend. It is structured as follows:

- Transaction Definitions
- Model Definitions.

The terms Application Program (AP), Resource Manager (RM), Transaction Manager (TM), and Communication Resource Manager (CRM) are defined in Section D.2 on page 83.

C.1 Transaction Definitions

Transaction

A *transaction* is a complete unit of work which, in general terms, can apply to many contexts. It may comprise many computational tasks including user interfaces, data retrieval, and communications. A typical transaction modifies resources. The model described in the referenced OSI TP standards defines the term *transaction* more precisely.

This guide refers to specific types of transaction, such as *global* and *distributed*. These are defined below.

Transaction Properties

Transactions typically exhibit the following properties:

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

Consistency which means that a completed transaction transforms a shared resource from one valid state to another valid state.

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

Durability which means 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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Global Transaction

The term *global transaction* collectively describes all the work done by participating RMs anywhere in the network for a single unit of work. An AP defines the start and end of a global transaction. A single, coordinating, TM manages its initiation and completion.

Distributed Transaction

This guide uses the term *global*, rather than *distributed*, transaction. Appendix D concentrates on global transaction processing whilst avoiding confusion about the actual location of the work and its distribution across physical locations, network nodes, or TM Domains.

Commitment

Commitment is the act that ends a transaction and makes permanent all changes to resources specified during that transaction.

Rollback

Rollback is the act that ends a transaction and nullifies or undoes all changes to resources specified during that transaction.

Transaction Completion

Transaction completion means either *commitment* or *rollback*.

Commitment Protocol

A *commitment protocol* is the synchronisation that occurs at transaction completion. The X/Open DTP model follows the *two-phase commit with presumed rollback*[2] protocol defined in the referenced OSI TP standards. A description of the basic protocol is given in Section D.4.3 on page 88. In certain cases, a global transaction may be completed *heuristically*. Heuristic transaction completion is described in Section D.4.5 on page 89.

RM Resource

In the X/Open DTP model, an *RM resource* is the collection of data that an RM manages. A resource may be *shared* with other global transactions or *dedicated* to a single transaction.

RM Native Interface

The RM's native interface is the RM-defined API by which an AP operates on the RM's resource. The RM may support a standard interface, such as SQL or ISAM, in which case the AP may be portable to other RMs that use the same interface. The RM may, on the other hand, offer a proprietary interface specific to its services.

---

2. To aid readability, some parts of this guide use the term *two-phase commit* as an abbreviation of *two-phase commit with presumed rollback*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

C.2 Model Definitions

Instance of the Model

An *instance of the model* (or *instance*)[3] is the set of computing entities that implement the functional components and interfaces of all or part of an application within the X/Open DTP model. Each *instance* may support one AP, one TM, and multiple RMs. A distributed application is represented by two or more *instances* and includes a CRM in each *instance.* An *instance* is part of a single TM Domain. A TM Domain can contain one or more instances.

The possible implementation of product structures that make up an instance may vary from simple to very complex. Appendix E addresses some frequently asked questions about how existing product structures fit within the X/Open DTP model.

TM Domain

A *TM Domain* consists of one or more instances that use the same TM. This TM is common to all applications that run within that TM Domain. The common TM uses logically-shared data structures and logs for global-transaction management, such as when issuing global transaction identifiers or when coordinating global-transaction recovery.

The figure below illustrates four *instances* of applications within a single TM Domain. The TM is the same in each *instance* and is labelled **TM1**. Different RMs may participate in each instance.

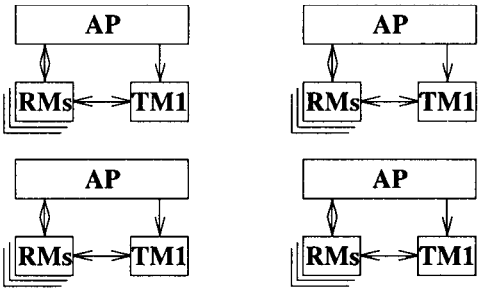


Figure C-1 A TM Domain with Four Instances

3. To aid readability, some parts of this guide use the term *instance* as an abbreviation of *instance of the model*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

X/Open DTP Model

At a basic level, the model is a software architecture that allows a single AP to share resources provided by multiple RMs within a single TM Domain, and allows RM-internal work to be coordinated into a global transaction using a single TM. It uses the TX and XA interfaces.

The full model is a software architecture that allows multiple APs to share resources provided by multiple RMs located across one or more TM Domains, and allows RM-internal work to be coordinated into a global transaction using multiple TMs. The full model uses the the STDL language, the TX, XA, and XA+ interfaces, the interfaces provided by the Communication Resource Manager (CRM) specifications, namely TxRPC, XATMI, and CPI-C Version 2, and the XAP-TP interface to coordinate communication through OSI TP.

Threads

A thread is the entity, with all its context, that is currently in control of a processor. For portability reasons, the notion of thread must be common among the AP, TM, and RM.

The thread concept is central to the TM's coordination of RMs. APs call RMs to request work, while TMs call RMs to delineate transaction branches. The way the RM knows that a given work request pertains to a given branch is that the AP and the TM both call it from *the same thread*. For example, an AP thread calls the TM to declare the start of a global transaction. The TM records this fact and informs RMs. After the AP regains control, it uses the native interface of one or more RMs to do work. The RM receives the calls from the AP and TM in the same thread.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Appendix D: DTP Model — The Model

(For informative purposes, this appendix provides a copy of Chapter 3 of the published X/Open Guide: Distributed Transaction Processing Reference Model Version 3.)

This chapter gives an abstract description of the X/Open DTP model of an application program environment for global transaction processing.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.1 Functional 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.

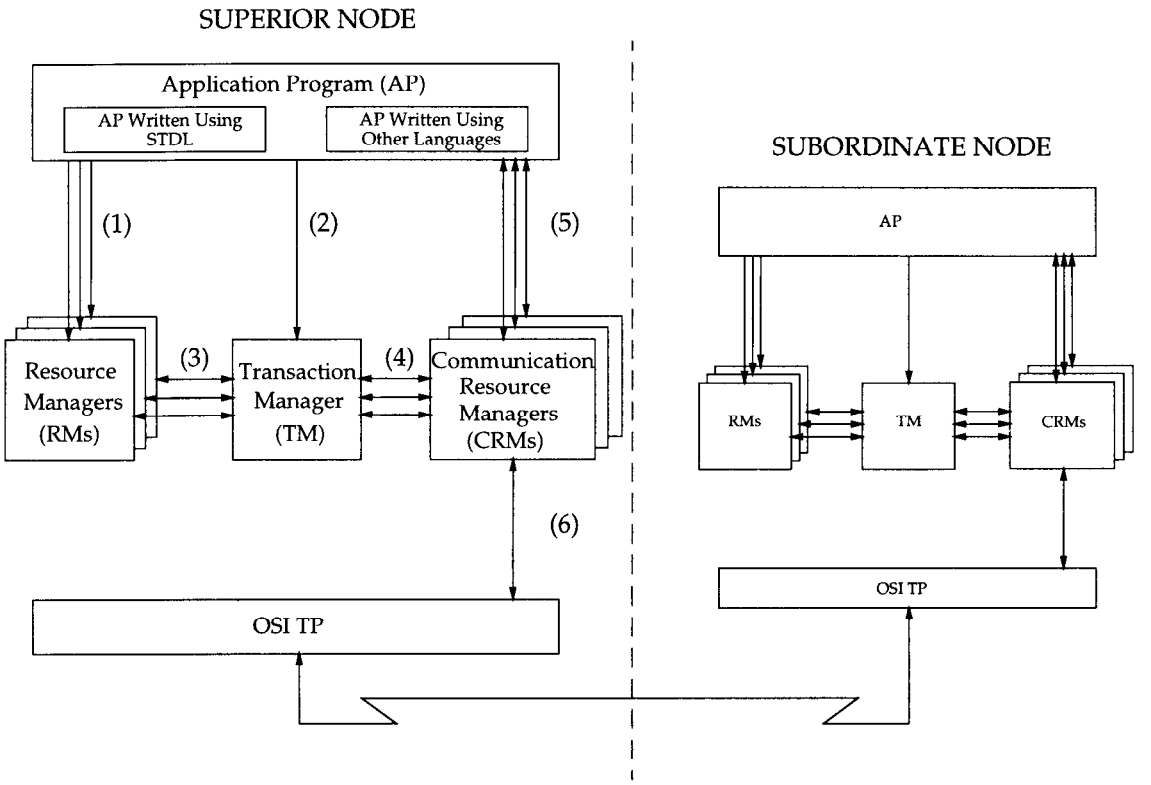


Figure D-1 Functional Components and Interfaces

Descriptions of the functional components shown can be found in Section D.2 on page 83. 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 D.3.1 on page 85. The subsequent sections provide a description of how the model works.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.2 Functional Components

D.2.1 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 Version 2 interfaces, described in Section D.7 on page 96.

Where two or more APs support a global transaction, one AP is designated the superior AP and the other AP(s) the subordinate(s). The superior AP makes distributed requests for services to be provided by subordinates, for example service requests using the XATMI interface. A superior AP can have many subordinate APs whereas each subordinate AP has only a single superior. A subordinate AP can also be superior to other APs working at a lower level. Hence a hierarchy of APs can be built up.

An AP acting as the superior typically exerts more control than any of its subordinates. For example, depending on the communication paradigm, only the superior AP may be allowed to start and commit a global transaction. Its subordinates however may roll back the global transaction.

APs can be written using the X/Open HTL, the X/Open APIs, or a combination of the two. See Section D.8 on page 98 for further information on the X/Open HTL.

D.2.2 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.

In addition, the following information applies where two or more TMs cooperate within a global transaction.

The TM that works on behalf of the superior AP is designated the superior TM. Other TMs are designated subordinate TMs. The superior/subordinate relationship is the same for TMs as for their respective APs. Within a global transaction, each TM only controls those RMs within its own instance of the model. Through their CRMs (see Section D.2.4 on page 84), TMs communicate global transaction information between each other by use of the XA+ interface.

D.2.3 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.

In addition, the following information applies when RMs take part in a global transaction involving two or more TM Domains or instance of the model.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Within the context of the X/Open DTP model, an RM receives global transaction information only from the TM in the instances of the model with which it is registered. This information is provided through the XA interface. An RM uses the *ax_*()* functions of the XA interface to communicate with its TM.

Outside the X/Open DTP model context, an RM is free to access data at many physical locations. For example a Relational Database Management System (RDBMS) may use a *gateway* technology to access a remote database. The RDBMS presents the AP with a single-image logical view of the database concerned, and the AP is not aware of either its physical location or the detailed structure of its components. In such a setup, the RM must be able to complete its own RM-internal work atomically, on behalf of the global transaction, no matter how physically diverse are its resources.

D.2.4 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 Version 2) 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*.

The CRM that supports the superior TM is called the superior CRM. Other CRMs, that support other TMs, within the global transaction, are called subordinate CRMs. The superior/subordinate relationship is the same for CRMs as for their respective APs and TMs. This means that a CRM can only directly support the AP and TM of its own instance. It follows that a CRM in one instance cannot directly support an AP and TM in another instance, but must request the services of the CRM for that instance. This paired relationship between CRMs is shown in Figure D-1 on page 82.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.3 Interfaces between Functional Components

D.3.1 Functional Component Interfaces

There are six interfaces between software components in the X/Open DTP model. The numbers correspond to the numbers in Figure D-1 on page 82.

(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 allows the AP to coordinate 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 **STDL** specification and the **TX (Transaction Demarcation)** specification[4].

In addition, the following information applies when two or more APs cooperate within a global transaction.

If the AP is designated as the superior, it uses the AP-TM interface to delimit global transactions. If the AP is designated as a subordinate, then depending on the communication paradigm employed, it may not be permitted to use all the facilities of the AP-TM interface. For example, the subordinate AP may not be allowed to request global transaction commitment.

A TM may receive global transaction information from another TM routed through remote and local CRMs. For example a superior TM may inform subordinate TMs to commit a global transaction without the subordinate APs' knowledge. Subordinate APs can however access information about the current global transaction context by calling *tx_info*( ).

(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[5]. 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.

The TX and XA interfaces cooperate to provide global transaction management between the AP and TM and, at a lower level, the TM and participating RMs. The TX interface drives the XA interface when managing global transactions. The XA interface reports back

4. The TX (transaction demarcation) interface provides both a C and a COBOL programming interface. TX functions are in lower-case for C, for example *tx_commit*( ) and in upper case for COBOL, for example TXCOMMIT. To aid readability, this guide uses the C versions only.
5. The XA interface is a C interface only, between the TM and RM. It is not accessed directly by the AP.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

to the TM the success (or otherwise) of global transaction work at RM level. The TM then collates the RM returned information and reports back to the AP the success (or otherwise) of global transaction management at TM level.

Since the XA interface is invisible to the AP, the TM and RM may use other methods to interconnect without affecting application portability.

(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 guide 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 **TxRPC** specification)
- The XATMI interface (see the **XATMI** specification)
- The CPI-C Version 2 interface (see the **CPI-C Version 2** specification).

These paradigms are described in more detail in Section D.7 on page 96.

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 the OSI TP standards.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.3.2 Data Interfaces

Transaction Identifier

A transaction identifier (XID) is a data structure that a TM assigns. It represents the unique relationship between an AP, the work it issues to RMs, and the global transaction which the TM manages on behalf of the AP.

The XID lets the TM track and coordinate all of the work associated with a global transaction. Each RM maps the XID to the RM-internal work it does for the global transaction. To ensure global uniqueness, the XID should contain *atomic action identifiers* as specified in the referenced OSI CCR standards. For more information on XIDs, see also the **XA** specification.

XA Switch Structure

Each RM provides a set of pointers to the functions that the TM calls, in a data structure known as the XA Switch structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.4 Activity Involving a Single AP

D.4.1 Transaction Initiation

When an AP instructs its TM to start a global transaction, the TM informs all participating RMs in order to associate with the global transaction any work the AP may request of them.

Some RMs are configured so that the TM does not inform them when a global transaction starts. Instead, the RM contacts the TM to become associated with a global transaction only after the AP calls it to request actual work. This is known as *dynamic registration*.

D.4.2 Transaction Association

A thread working on a transaction branch may suspend association with that transaction branch when blocking, awaiting a message from another application. The TM may resume the association in the same thread or optionally in a different one subject to RM control. The TM suspends an association by issuing *xa_end*( ) to the local RMs with the TMSUSPEND flag set. The TM also sets the TMMIGRATE flag if it might resume the association in a different thread. If all RMs permit migration, the TM may resume the association in the same or a different thread. Otherwise, the association must be resumed in the same thread. Each RM must retain transaction context (for example locks and cursors) while the association is suspended.

D.4.3 Transaction Commitment

When an AP instructs its TM to commit a transaction, the TM and participating RMs use the two-phase commit protocol to ensure transaction atomicity. The AP requests that the TM initiate commitment.

In Phase 1, the TM issues *xa_prepare*( ) which requests each participating RM to *prepare to commit* its work and report whether it can guarantee its ability to commit the work it did on behalf of the global transaction. If an RM can commit its work, it replies affirmatively. A negative reply reports failure for any reason.

In Phase 2, the TM directs all RMs either to commit or roll back the work done on behalf of the global transaction. Whether the TM issues *xa_commit*( ) or *xa_rollback*( ) depends on the RMs' responses during Phase 1. All RMs commit or roll back changes to shared resources and then return status to the TM.

When an AP requests commitment of a global transaction, the TM reports back to the AP whether commitment or rollback was the outcome. This is based on reports the TM received from all participating RMs.

The **XA** specification contains two optimisations in the calling sequence between TM and RM:

- An RM can withdraw from further participation in a global transaction during Phase 1 if it was not asked to update resources. This is known as *read-only optimisation*.
- A TM can use *one-phase commit* if it is dealing with only one RM that is making changes to resources.

The **XA** specification discusses requirements for the stable recording of transaction data, and specifies when the TM and participating RMs are free to discard their knowledge of the global transaction. See the referenced **XA** specification for further details.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.4.4 Transaction Rollback

Transaction rollback can occur in three ways:

- **Explicit Rollback**
  The AP requests explicit rollback of the global transaction.
- **Implicit Rollback**
  The TM rolls back the global transaction if any RM responds negatively to the Phase 1 request.
- **Presumed Rollback**
  On restart, an RM rolls back any transaction that was active at the time of failure, provided that it has not already responded positively to a Phase 1 *prepare to commit* request from the TM.

In each case, the participating RMs must not allow any changes to resources to become permanent.

D.4.5 Heuristic Transaction Completion

In certain unusual cases, an RM may unilaterally commit or roll back changes to recoverable resources that it made within a global transaction. This may happen, for example, when the RM experiences too long a delay between Phases 1 and 2 of the two-phase commit protocol, or it is prompted by operator intervention to complete (commit or rollback) its RM-internal transaction. In such cases, the RM is said to make a *heuristic* decision. Such a decision is made by the RM without knowledge of the state of the global transaction of which it is part. The RM may then unlock *shared* resources and allow other global transactions to make further changes.

If a heuristic decision is taken, it is possible that at a later point the global transaction may complete with a decision contrary to that made by the RM. In such a circumstance, the global transaction fails to maintain global **ACID** properties. In other cases a TM may not be able to determine whether an RM's heuristic decision matched the TM's global transaction completion decision. If a contrary heuristic decision is reported by an RM, the TM reports to the AP that a heuristic decision has occurred. This decision is either contrary to the AP's direction or both complementary and contrary *(mixed)* depending on the combined results from all participating RMs. If the TM cannot determine whether an RM's heuristic decision matches the decision made by the TM, the TM reports to the AP that there is a possibility *(hazard)* of a contrary heuristic decision.

In the X/Open DTP model, an RM that reports heuristic completion to the TM must not discard its own knowledge of the decision until authorised by the TM. Until this happens, a TM can request *xa_recover*( ) of an RM to collect a list of transaction identifiers for heuristically completed and prepared transactions.

The referenced OSI CCR standards define heuristics more precisely.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.4.6 Recovery after Failure

Recovery is the process of restoring resources to a consistent state after various types of failure. Recovery processing in an X/Open DTP system is compatible with that described in the OSI TP standards, which define the presumed-rollback protocol. The X/Open DTP model makes the following assumptions:

- that the TM and RMs have access to stable storage
- that the TM initiates and controls transaction recovery
- that RMs provide for their own restart and recovery as directed by the TM.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.5 Distributed Communication Facilities

D.5.1 Communication within TM Domains

Distributed applications that support global transactions across two or more instances of the model may communicate with each other within a single TM Domain. Communication between two or more instances is managed at each end by a CRM. The RMs coordinate with the domain's TM used in the appropriate instance.

D.5.2 Communication across TM Domains

The X/Open DTP model specifies that heterogeneous TM Domains support CRMs that use the OSI TP protocol as a common communication layer. Proprietary protocols or the OSI TP protocol may be used between homogeneous TM Domains.

D.5.3 Sharing Resources across TM Domains

TM Domains may share resources in ways other than by using CRMs. For example, RDBMSs in different TM Domains may share a common back-end database. In this case, an RM may receive XIDs from several TMs that have no knowledge of each other. Use of a common format of XID, and adherence to the rules for uniqueness of the global transaction identifier, (see Section D.3.2 on page 87), ensure that the RM can distinguish between different global transactions and their associated transaction branches.

D.5.4 Global Transaction Demarcation

The AP-TM interface is used in the X/Open DTP model application program environment to define global transaction start, end, commitment or rollback and other facilities. Before an AP starts a global transaction, it is in non-global transaction status, and so any work requested of RMs by the AP during this period is not part of a global transaction started later.

The TX interface supports the concept of *unchained* and *chained* global transactions. In unchained mode (which is the default), when a global transaction completes, a new transaction does not automatically start until the AP calls *tx_begin*( ) again.

When in chained mode, after the initial global transaction starts and completes, the next global transaction is started automatically.

D.5.5 Global Transaction Tree Structure

Within the X/Open DTP model, global transactions that operate across distributed APs are managed by use of a tree structure. This is shown in Figure D-2 on page 92.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

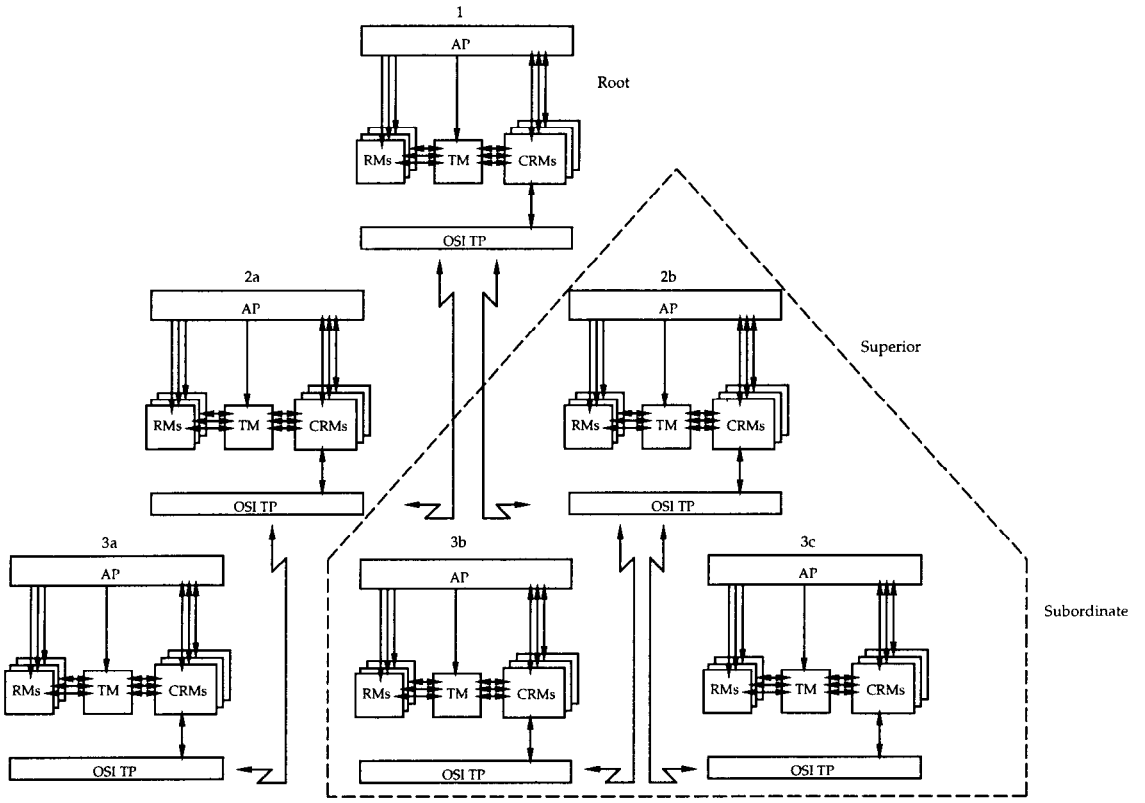


Figure D-2 Global Transaction Tree Structure

When distributed communication occurs between two instances, they have a relationship of *Superior* and *Subordinate*. The instance that requests the participation of another instance in a global transaction is called a superior. The requested instance is called a subordinate. Especially, an instance that does not have a superior is called a root.

An example of this is shown in Figure D-2, where 2b is a superior to both 3b and 3c; 2b, however, is a subordinate to 1.

D.5.6 Global Transactions and the Transaction Tree

A global transaction has one or more transaction branches. Each branch represents part of the work in support of a global transaction for which a TM and set of participating RMs engage.

During two-phase-commit processing, the superior TM manages commitment coordination with subordinate TMs and their participating RMs at branch level. This is achieved by the superior CRM reporting back to the superior TM the success or failure of one or more branches in response to prepare and commit requests.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.5.7 Tightly- and Loosely-coupled Threads

Many application threads can participate in a single global transaction. All the work done in these threads is atomically completed. Within a single global transaction, the relationship between any pair of participating threads is either *tightly-coupled* or *loosely-coupled*:

- A tightly-coupled relationship is one where a pair of threads are designed to share resources. In addition, with respect to an RM's isolation policies, the pair are treated as a single entity. Thus, for a pair of tightly-coupled threads, the RM must guarantee that resource deadlock does not occur within the transaction branch.
- A loosely-coupled relationship provides no such guarantee. With respect to an RM's isolation policies, the pair may be treated as if they were in separate global transactions even though the work is atomically completed.

Within a single global transaction, a set of tightly-coupled threads may consist of more than just a pair. Moreover, many sets of tightly-coupled threads may exist within the same global transaction and each set is loosely coupled with respect to the others.

D.5.8 Commitment Coordination

The TM that manages global transaction completion is called the *commitment coordinator*. It is the AP of the commitment coordinator instance of the model that requests commitment.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.6 Activity Involving Two or More APs

In addition to the description of transaction management given in Section D.4 on page 88, the following information applies when two or more distributed APs cooperate within a global transaction.

D.6.1 Transaction Initiation

When an AP makes a request to a remote AP for the first time, a new transaction branch for a subordinate must be created. There are two methods of creating and managing branches. One is called *TM-managed transaction branches*, the other is called *CRM-managed transaction branches*.

With TM-managed transaction branches, the CRM local to its instance of the model requests the TM to build and log details of a new transaction branch prior to accessing the remote AP for the first time.

With CRM-managed transaction branches, the CRM builds the transaction branch and manages it without involvement from the TM. Such a CRM appears to the TM as an ordinary RM, but in effect is acting as the distributed manager of the TM in the global transaction.

In either case, subsequent accesses to the same remote AP are made through the same transaction branch. A CRM accesses different remote APs through different transaction branches.

The CRM uses transaction branches to target global transaction co-ordination information passed on to the (subordinate) branch via XA+ commands issued by the TM.

A CRM can be configured so that the TM does not inform it when a global transaction starts. Such a CRM contacts the TM to become associated with a global transaction only after the AP calls it to make a distributed request. This is known as *dynamic registration*.

D.6.2 Transaction Association

The CRM suspends the association of a thread with a transaction branch by issuing *ax_end*( ) to the TM with the TMSUSPEND flag set. The CRM also sets the TMMIGRATE flag if it might resume the association in a different thread. If the TM and all local RMs permit migration, the CRM may resume the association in the same or a different thread. Otherwise the association must be resumed in the same thread. (See Section D.4.2 on page 88).

D.6.3 Transaction Commitment

When an AP in the root or superior instructs its TM to commit a global transaction, the TM follows the two-phase commit protocol. In addition to Phase 1 coordination of commitment of participating RMs, the superior TM uses its CRM to instruct all other transaction branches within the global transaction to prepare to commit. The CRM passes an *xa_prepare*( ) request to subordinate TMs in other branches. The CRM then waits for and returns to its own TM each transaction branch response.

If the TM receives a positive response from its participating RMs and from all associated transaction branches, the TM initiates Phase 2 by requesting *xa_commit*( ) of the global transaction. The TM again uses its CRM to instruct all other transaction branches and await their response.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.6.4 Transaction Rollback

If the TM receives a negative response from a local RM or from any of the associated transaction branches, the TM initiates Phase 2 by requesting *xa_rollback*( ) of the global transaction. The TM again uses its CRM to instruct transaction branches and await their response.

D.6.5 Heuristic Transaction Completion

When a global transaction extends across two or more transaction branches through communication, RMs in any of the branches of subordinates might experience too long a delay between Phases 1 and 2 of the two-phase commit protocol. For example, a communication failure might have delayed the commit/rollback indication from the superior TM. As Section D.4.5 on page 89 describes, a subordinate RM may choose to complete its work without the *xa_commit*( ) instruction from the TM.

During Phase 2 of the two-phase commit protocol, the superior TM receives a response from a transaction branch that heuristic completion of a subordinate RM has occurred. The superior TM logs the response and informs the superior AP.

In some cases, for example after a communication failure, the superior TM may not be able to determine whether heuristic completion of a subordinate RM has occurred. The superior TM logs the fact and informs the superior AP.

D.6.6 Recovery after Failure

Recovery processing in an X/Open DTP system is described in Section D.4.6 on page 90. In addition, when a global transaction extends across two or more transaction branches, a CRM may request *ax_recover*( ) of a TM to collect a list of transaction identifiers for prepared transactions for which the TM has information logged on behalf of the CRM.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.7 CRM Communication Paradigms with APs

D.7.1 The TxRPC Interface

For distributed applications using the remote procedure call (RPC) mechanisms of the X/Open Distributed Computing Services (XDCS) Framework, a Transactional Remote Procedure Call, TxRPC, interface is offered. This allows application writers to invoke remote procedures in the same form as local procedures. Typically the calling environment is an AP referred to as the *client*. The environment where the call is processed is referred to as the *server*. The client calls a remote procedure, and waits for the results of the call. When the server finishes processing the procedure, it returns the results to the client which then resumes execution. The TxRPC interface optionally permits the AP to extend the scope of a global transaction from the client to the server.

Calls that have either *transaction-mandatory* or *transaction-optional* attributes are called *TxRPC operations*. If a TxRPC operation takes place within the scope of a global transaction, the call becomes a *transactional RPC*.

Descriptions of the types of parameters used in the call, plus additional global transaction information semantics are available to the client and server APs via an *Interface Definition Language (IDL)*.

For details of this interface, see the referenced **TxRPC** specification.

D.7.2 The XATMI Interface

For distributed applications using service requests in a client-server paradigm, X/Open offers the XATMI interface. This interface supports the use of *client* APs requesting services to be performed. A *service* is an AP routine that performs a specific application function on behalf of the client. The structure of the client AP is defined entirely by the application writer and the structure of the service routine is defined by the XATMI interface.

XATMI supports two types of service:

Request/response

These services receive a single request from a client AP and produce at most a single response to the request. Requests can be made in two ways:

- The client AP can make a synchronous request by calling *tpcall*( ). The client AP is then suspended until a response is received.
- The client AP can make an asynchronous request by calling *tpacall*( ). The client AP can then continue work while the service routine processes the request. The client AP may, in fact, makes other requests and so exploit parallelism since multiple requests can be simultaneously processed by different service routines. The *tpacall*( ) function also returns a *call descriptor*. This is used later as a parameter by the client AP when it calls *tpgetrply*( ) to receive a service response to a specific earlier request.

Conversational Services

These services are invoked by a *tpconnect*( ) request from the client AP. Once the connection is established and the service invoked, the client and service can exchange data using *tpsend*( ) and *tprecv*( ) for an indefinite period of time in a conversational manner. Conversations take place in *half duplex* mode, that is only one AP can send data at a time. XATMI does not allow the receiver AP to send data until the sender AP yields its control of the conversation. The connection request is completed when the service routine calls *tpreturn*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

For details of this interface, see the referenced **XATMI** specification.

D.7.3 The CPI-C Version 2 Interface

For distributed applications using a conversational paradigm, where communication takes place through an application-defined exchange of messages, X/Open offers the CPI Communications (CPI-C) Version 2 interface[6] in addition to the XATMI interface (described above).

The conversational model of program-to-program communication is commonly used in the industry today, and a wide variety of applications are based on this model. The model is historically thought of in terms of two applications *speaking* and *listening*, hence the term *conversation*. A conversation is simply a logical connection between two programs that allows the programs to communicate with each other. From an application's perspective, X/Open CPI-C Version 2 provides the function necessary to enable this communication.

The CPI-C conversational model is implemented in two major communication protocols: Open Systems Interconnection Distributed Transaction Processing (OSI TP)[7] and Advanced Program-to-Program Communications (APPC). The APPC model is also referred to as logical unit type 6.2 (LU 6.2). X/Open CPI-C Version 2 provides access to both communication protocols.

A primary benefit of this design is that CPI-C Version 2 defines a single programming interface to the underlying network protocols across many different programming languages and environments. The interface's rich set of programming services shields the AP from details of system connectivity and eases the integration and porting of the application programs across the supported environments.

Note: The X/Open **CPI-C Version 2** specification derives from the CPI-C 2.0 specification[8] produced by the CPI-C Implementers' Workshop, but with the following major differences:

- X/Open CPI-C Version 2 only supports the C and COBOL programming languages.
- X/Open CPI-C Version 2 does not support the concept of a *distributed directory*.

D.7.4 Relationships between the Communication Paradigms

Where different applications use the *same CRM communication paradigm*, X/Open facilitates the following goals of paradigm consistency:

- Applications are portable from one TM domain to another.
- Different TM domains can interoperate within the same communication paradigm by using the OSI-TP protocol.
- If the XA+ interface is used, different CRM implementations are interchangeable (this feature is optional).

Relationships between *different CRM communication paradigms* are not determined by the model.

---

6. This specification supersedes the X/Open **Peer-to-Peer Snapshot**, published in 1992, as one of the three X/Open interfaces between an AP and a CRM.
7. See the referenced OSI TP standards.
8. See the referenced CIW CPI-C specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D.8 High-level TP Language

X/Open has adopted a high-level TP language (HTL) called the Structured Transaction Definition Language (STDL). The purpose of the XHTL is to allow an AP to be written at a higher, language-syntax, level than the relevant X/Open APIs (namely the TX and CRM APIs) which take the form of embedded services. The XHTL is intended to be mappable to all of the X/Open CRMs in order to provide CRM- and communication-paradigm independence. The **STDL** Preliminary Specification is mapped only to the TxRPC CRM.

An AP writer can use the XHTL instead of the relevant X/Open APIs to specify a sequence of operations that involves resources such as databases. The AP writer can use STDL syntax in place of the TX interface to coordinate global transaction management. An AP writer can also use STDL syntax in place of the TxRPC interface for AP to AP communications.

STDL implements transactional operations within higher-level syntax such as C or COBOL. AP writers can choose to use STDL when they want to work at the language-syntax level, which provides the benefit of syntax checking, potentially resulting in fewer runtime errors. Using STDL, the AP writer creates separate STDL procedures that contain the transactional operations and that call C, COBOL, or possibly other language procedures for other operations. STDL procedure calls are transactional when the called procedures access RMs, and are optionally transactional when calling remote procedures that access RMs.

STDL supports both non-transactional and transactional modes (chained or unchained).

See the referenced **STDL** specification for further information about STDL syntax and its relationship to the syntax of other languages such as SQL, C, and COBOL, as well as the definition of RM access and mapping to the TxRPC protocol.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Appendix E: DTP Model—Frequently Asked Questions

(For informative purposes, this appendix provides a copy of Appendix A of the published X/Open Guide: Distributed Transaction Processing Reference Model Version 3.)

How do existing products' structures compare with the model's structure?

See the questions below.

What elements in the model lead to a Transaction Processing Monitor?

A Transaction Processing Monitor may often include both a TM component and a CRM component. It may also provide additional features outside the X/Open DTP model such as task scheduling and access to security subsystems.

How does the model handle non-global transactions?

The X/Open DTP model does not insist that an AP performs all work within the scope of a global transaction. Work performed when no global transaction is defined is termed a *non-global transaction*. In non-global transaction processing, one or more RMs are used by the AP, but are not coordinated by the TM. RM-specific work may be committed using the appropriate native interface commands, such as SQL COMMIT. As far as individual RMs are concerned, the work performed is transactional and the **ACID** transaction properties are obeyed, but when two or more RMs are involved, there is no coordinated commitment between them.

An AP may choose to run *non-global* and *global* transactions within the same TM Domain (see **TM Domain** on page 79 for a definition). The two approaches for such a combination are:

- The AP runs non-global and global transactions in parallel.
- The AP switches between non-global and global transactions, but never runs both types at the same time.

In either case, the coordination of the global and non-global transactions is an additional issue of application design. Most applications should not need to mix non-global and global transactions.

What are RM-internal transactions?

Many RM products structure their own work into transactions. An *RM-internal transaction* is a recoverable unit of work owned by a single RM. In the X/Open DTP model, a global transaction consists of one or more RM-internal transactions. The TM coordinates the start and completion of the RM-internal transactions of each participating RM.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

What elements in the model lead to a distributed database?

In a distributed database, the database RM manages the coordination of that part of the transaction it has internally distributed. Therefore, in the model it is a single RM component. When it receives the transaction instructions from the TM, for example a request to *prepare*, it transparently prepares any subordinates before responding positively to the TM's *prepare* request.

Where do distributed file systems fit?

A distributed file system can access data at multiple sites, but in general it is not transactional. That is, changes made to such a file system are not coordinated atomically with other actions in the transaction. Such a file system is outside the model. However, it is certainly possible for a distributed file system to be implemented with transactional semantics and to support the XA interface, in which case it would be an RM.

Can I use the model for distributed systems management?

The X/Open model can be used as a transactional framework for distributed system management. A CRM that implements CMIP or SNMP for example could, in conjunction with an X/Open TM, provide transactional system management with atomicity across the various objects and sites. The system management CRM could do the transaction management entirely on its own, or it could use the services of existing RMs to provide ACID storage properties.

Must *commit* occur in the same process that operated on the resources?

The *XA* specification specifically allows the TM to commit a transaction in a different thread of control from that which operated on the particular resource. In addition, the TM itself may have auxiliary processes to optimise the commit process, and so may the RM.

How does a client with just an AP and communication fit into the model?

Each instance of the X/Open DTP model must include a TM to provide transaction semantics. A client, operating without a TM, may send requests into a system that implements the model, but such a client is not part of any global transaction. If such a client has local resources, updates to them are not coordinated by the system.

How does a database client-server architecture fit into the model?

A client-server database that supports the XA interface is an RM within the model. As the next question elaborates, the process structure is not identical to the model components. The client process, for example, may be the AP plus libraries from the RM and TM, with the server process just an internal part of the RM.

A client-server database that does not support the XA interface for transaction control is outside the scope of the model.

What is the relationship between model components and processes?

X/Open model components (APs, TMs, RMs, and CRMs) in no way imply a particular process mapping. There are several reasons why a single component may be instantiated with multiple processes, or that multiple components may be collapsed into a single process:

- **Performance**

  A component may have separate processes to handle tasks somewhat asynchronous to the application logic, for example system management and logging.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- **Internal distribution schemes**

  For example, an RM may be a database with a client-server process model. Such an RM would be expected to provide the correct distributed semantics for its non-visible subordinates.

- **Practicality of linking**

  The model component called the AP needs to communicate with the TM and each RM, and, as such, must have within its process at least some library entry points that give it access to the TM and RM functions. As described above, the TM and RM functions may result in actions being taken by a different process, but the AP needs at least a call-level linkage to them.

An example of how the model might be projected onto processes is shown in the following figure, with ellipses indicating processes and rectangles indicating model components.

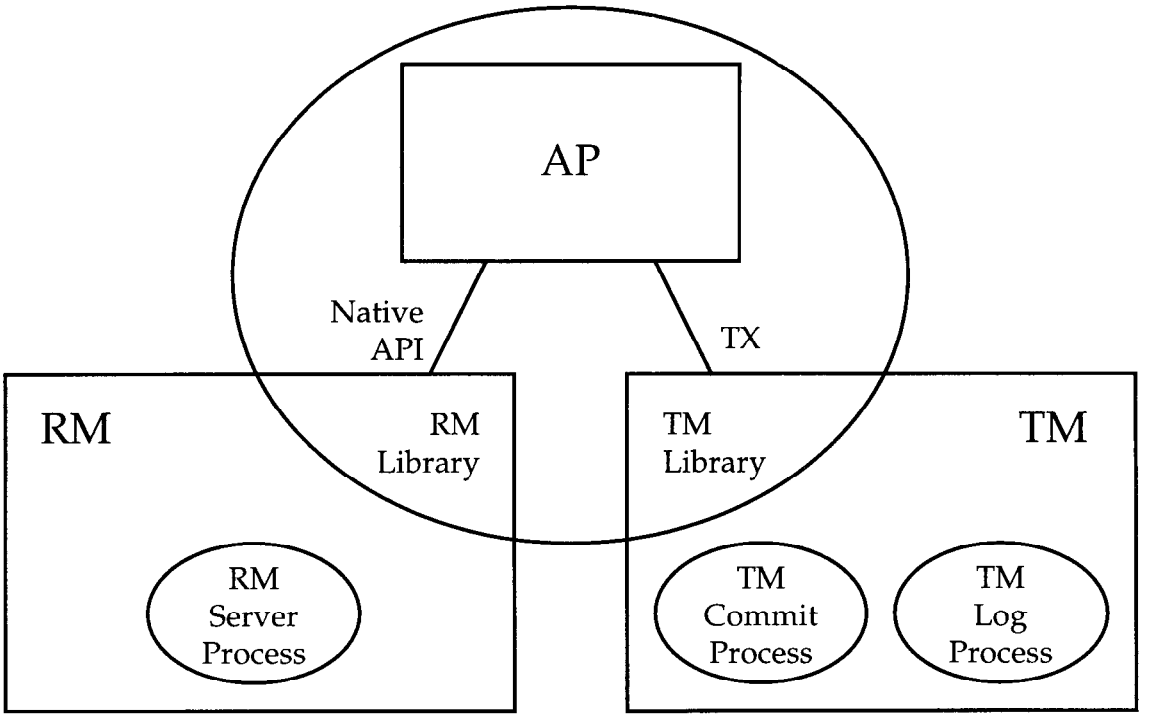


Figure E-1 Projection of Model onto Processes

Is an X/Open TM a Transaction Monitor?

An X/Open TM is a transaction manager, and as such coordinates the atomic completion of a transaction. A transaction monitor (or transaction processing monitor) typically supplies additional services such as distributed communication, task scheduling, and other facilities. An X/Open TM should be thought of as a small, though important, subset of a transaction monitor.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

How can I integrate existing products and services with products based on the XTP model?

There are two groups of existing products to consider: the system-level components of the model (TMs, RMs, and CRMs) and the products that provide services to those components (security subsystems for example).

The X/Open model concentrates on transaction atomicity over distributed products and sites. The model does not preclude the incorporation of existing products. In fact, because the X/Open model allows for native APIs to the RMs, products with existing APIs can be included. A product being incorporated as an RM or TM must have a *two-phase commit* ability that is compatible with the X/Open model. Also, for an existing product to interoperate with X/Open components, it must provide gateway software between the existing system and the XTP system. If these requirements are met, then the existing product can participate in X/Open transactions.

There are also existing products that supply complementary functions such as security checking or user interface capability. These can also be incorporated. They do not directly support transaction atomicity, but they provide additional features to make a more comprehensive application environment.

Finally, it is possible to map both the X/Open HTL and APIs to existing products to provide additional portability in existing product environments.

How do products implementing the model interoperate?

In terms of system component integration, products that implement the XA or XA+ interfaces can interoperate. In terms of network interoperability, two products that implement the same communication paradigm can interoperate, but two with different communication paradigms cannot. For example, some requests generated by the CPI-C paradigm have no equivalent meaning in TxRPC.

Can I switch easily from one vendor using the model to another?

The model allows for different native interfaces to RMs, and in general you can only switch between RMs that provide the same interface. For example, program recoding would be necessary to move from an RDBMS RM to an ISAM RM, or to move from a TxRPC CRM to an XATMI CRM. However, it is the intent of X/Open that programs be portable between vendors supplying the same RM or CRM API.

Why are there three communication APIs?

The three communication APIs currently supported were developed in response to both user requirements and technology divergence. For example, many users may not require the precise control afforded by the CPI-C paradigm, but applications needing to interact directly with legacy systems may.

Can the HTL and CRM APIs be combined in the same AP?

Yes, for example the combination of STDL and CPI-C in a single AP is possible, although the precise behavior of such a combination is not defined.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Appendix F: Bibliography

a. ISO/IEC 9075:1992, *Information technology—Database languages SQL.*

b. X/Open[9] Developers' Specification, August 1990, (ISBN: 1-872630-03-0, D010): *Indexed Sequential Access Method (ISAM).*

c. X/Open Preliminary Specification, December 1995, (ISBN: 1-85912-120-9, P536): *Structured Transaction Definition Language (STDL).*

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

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

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

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

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

i. X/Open CAE Specification, October 1995, (ISBN: 1-85912-130-6, C506): *Distributed Transaction Processing: The XATMI Specification.*

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

k. X/Open Guide, November 1992, (ISBN: 1-872630-64-2, G212): *Distributed Computing Services (XDCS) Framework.*

l. X/Open Guide, 1996, February 1996, (ISBN: 1-85912-170-5, G504): *Distributed Transaction Processing: Reference Model Version 3.*

m. IBM Corporation, (Order Number G325-0055): *Advanced Peer to Peer Comunications, Resource Reference.*

n. IBM Corporation, June 1994, (Order Number SC31-6180-01): *Common Program Interface Communications Specification, Second Edition (CPI-C 2.0).*

9. X/Open documents can be obtained from the X/Open Company Ltd., Apex Plaza, Forbury Road, Reading, Berkshire, RG1 1AX, United Kingdom (or by Electronic Mail to: XoSpecs@xopen.co.uk).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Index

ability to commit 8
absence of expected reply 9
access to resources 1, 5, 75
access to storage 90
account verification 5
ACID properties 77, 89, 99
  atomicity 77
  consistency 77
  coordination by TM 77
  durability 77
  isolation 77
  responsibility of RM 77
activity, functional components 88, 94
address space 7
addressing correct RM 69
administration not addressed 76
administrative action 89
administrative procedures 21, 65
ANSI C 19
AP 1, 75
  calls to commit 88
  component 83
  construction of 76
  CRM 86
  dissociating from RM 13
  environment 82
  instance of 5
  requirements 65
  subordinate 83
  superior 83
AP initiation of interface activity 11
AP to AP communication
  HTL 98
  STDL 98
AP-CRM interface 86
AP-RM interface 85
AP-TM interface 85
API
  comparison with 1
  portability 75
  RM 78
application
  communication 2, 75
  distribution 2, 75
  portability 75
  program 1, 75
application program 5, 83
  calls to commit 88
  component 83
  construction of 76
  environment 82
  interface to CRM 86
  interface to RM 85
  interface to TM 85
  sharing resources 1, 75
  subordinate 83
  superior 83
areas not addressed 76
association 88, 94
  of threads, state table 59
  of transactions 66
  suspending 14
asynchronous
  calling mode 17
  operations, state table 63
asynchrony
  RM option 67
  TM option 69
  use of RM flag word 21
atomic action identifier (OSI CCR) 19, 87
atomic commitment 6
atomicity 77, 88, 94
  at risk 89
  TM 83
autonomy of RMs 6
awareness, lack of between RMs 6
ax_ prefix 19
ax_ routines 12
ax_reg( ) 12, 15, 28
ax_unreg( ) 12, 15, 31
benchmarks not addressed 76
benefits 76
blocking 88, 94

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

blocking control thread....17
branch....88, 94
branch ID
component of XID....19
RM option....67
byte exchange in communicating XID....66
calling protocol
RM requirement....66
calling sequence....57
changes
making permanent....16
client-server architecture....100
CMIP....100
commit
coordination....93
decision....83
decision to....6
definition....78
guaranteeing ability to....8
one-phase....9, 88
prepare to....8, 16
protocol definition....78
two-phase....88, 94
COMMIT WORK statement in SQL....68
commitment protocol....8
alternate....9
disruptions from error....9
optimisations of....8
phases of....8
RM requirement....66
state table....61
commitment, atomic....6
committing transactions....5, 16
Common Usage C....19
communication....1
across TM domains....91
commit coordination....93
CRM with AP....96
distributed facilities....91
global transaction demarcation....91
global transaction tree structure....91-92
sharing resources across TM domain....91
technique for RM....76
within TM domain....91
communication protocol....2, 75
communication resource manager....2, 75
communication with AP....96
component....84
dynamic registration....94
heuristic....95
heuristic decision....95
independence from STDL....98
interface to AP....86
interface to OSI-TP....86
interface to TM....86
relationship between paradigms....97
subordinate....84
superior....84
completion....88
coordinate....83
heuristic....9, 89, 95
testing for....17
completion of transactions....5, 16
component....4, 82
activity....88
activity between....94
AP....1, 75, 83
AP-CRM interface....86
AP-RM interface....85
AP-TM interface....85
CRM....2, 75, 84
CRM-OSI TP interface....86
failure....83
interchangeability....75
interfaces between....85
interoperability....75
relationship with processes....100
RM....1, 75, 83
RM-TM interface....85
TM....1, 75, 83
TM-CRM interface....86
computational task....5
concluding involvement....8
concurrent access to RMs....66
configuration file....13
editing....13
configuration not addressed....76
consensus....76
consistency....77
consistent effect of decisions....5
consistent state....5, 90, 95
context....7
control....4, 82
control returned to caller....17
control, thread of....7
cooperating unit....76
coordination of transactions....6
copying XID....20
CPI-C Version 2 interface....80, 83-84, 86, 97
CRM....2, 75
communication with AP....96
component....84

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

dynamic registration 94
heuristic 95
heuristic decision 95
independence from STDL 98
relationship between paradigms 97
subordinate 84
superior 84
CRM independence
HTL 98
STDL 98
CRM-AP interface 86
CRM-OSI TP interface 86
CRM-TM interface 86
custom software, eliminating 76
data structure
XID 87
database 1, 75
distributed 100
database management system 5
DBMS 5, 83
atomic commitment 5
decision
to commit 6
to commit or roll back 5
uniform effect 5
decision to commit 83
declarations, changing 21
definition
commit protocol 78
distributed transaction 78
DTP model 4, 79-80, 82
global transaction 78
heuristic outcome 78
instance of the model 79
non-global transaction 99
RM native interface 78
RM resource 78
RM-internal transaction 99
TM domain 79
transaction 77
transaction commit 78
transaction completion 78
transaction properties 77
two-phase commit 78
definitions 5
delivered products 65
demarcation of transaction 83
discarding knowledge of transaction 8
dissociation
from RM 13
from transaction 14
of threads 16
of threads, state table 59
distributed database 100
distributed file system 100
distributed RM 76
distributed system management 100
distributed transaction 5
definition 78
distributed transaction processing 5
DTP
definition of 5
implications of 5
DTP model 1, 4, 75, 82
definition 4, 80, 82
DTP Model
instance 79
durability 77
dynamic registration 15
of RM 15
RM option 67
state table 59
use of RM flag word 21
dynamic registration CRM 94
dynamic registration RM 88
ending involvement, dynamic RMs 15
entry in state table 58
entry points, pointers to 21
error versus veto 14
EXEC SQL COMMIT WORK 68
EXEC SQL ROLLBACK WORK 68
executable modules, linking 21
existing products 102
expected reply, absence of 9
explicit rollback 89
failure 9, 90, 95
after prepare 16
component 83
correctable 9
locally-detected 9
of system component 5
recovery 6, 17
to prepare to commit 8
FAQ (frequently asked questions) 99
file access method 83
basis for RM 5
file access system 1, 75
file system
distributed 100
flag
absence of 22
definitions 22

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

naming....19
word....21
word for RM....21
flow of control....4, 82-83
forgetting transaction....16, 88
formatID....20
forms management not addressed....76
frequently asked questions....99
functional component
AP....83
CRM....84
RM....83
TM....83
functional model....4, 82
global transaction....83
definition....78
demarcation....91
tree structure....91-92
with non-global....99
global uniqueness of XID....19
guarantee
commit....88
rollback....88
guaranteeing ability to commit....8
guaranteeing global serialisability....8
header file....19
heterogeneous
environment....76
heuristic....89
decision....89, 95
transaction completion....89, 95
heuristic completion....68
in state table....62
RM requirement....66
heuristic decision....9, 16
heuristic decisions
affected transactions....17
matching....9
notification of....9
heuristic outcome
definition....78
HTL....75
AP to AP communication....98
CRM independence....98
language....83, 98
language-syntax level....98
managing global transactions....98
mapping to CRM....98
procedures....98
RM access....98
transactional....98
identification, of calling thread....66
immediate return mode....17
implementation requirements....65
implementor
options for RMs....67
requirements for TMs....69
implications of DTP....5
implicit rollback....89
incomplete state of transaction....14
index to services....12
information
on transactions....12
RM requirement to publish....67
string....67
informational string....67
locating....69
informing RMs....88, 94
initialisation
of RMs....13
string....13, 67
string, locating....69
initiation
of completion....16
of interface activity....11
initiation of transaction....94
initiator name (OSI CCR)....20
instance
of AP....5
instance of the model
definition....79
interchangeability....75-76, 102
interchangeability, ensuring....24
interface....4, 82-83
activity, initiation....11
AP-CRM....86
AP-RM....85
AP-TM....85
between components....85
CPI-C Version 2....80, 83-84, 86, 97
CRM-OSI TP....86
data....87
function....85
illustrated....4, 82
ISAM....78, 83, 85
mapping....85
native....68
overview....11
proprietary....78
RM-TM....1
SQL....78, 85
system-level....1, 75

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TM-CRM 86
TM-RM 85
TX 80, 85, 91
TxRPC 80, 83-84, 86, 96
XA 80, 84-85, 88
XA+ 80, 83-84, 86, 94
XAP-TP 80, 84, 86
XATMI 80, 83-84, 86, 96
international standards 76
interoperability 75-76, 102
involvement
concluding 8
in transaction 8
of RM 15
ISAM 83
basis for RM 5
concept of transactions 68
interface 78, 85
ISO object identifier 69
isolation 77
joining transaction 14
knowledge of transaction
discarding 8
lack of update 8
language
HTL 75, 83, 98
STDL 80, 98
language-syntax level
HTL 98
STDL 98
linking 21, 65, 100
assumptions by TM 69
information 67, 69
skeleton routines 65
local failures 9
local recovery done by TM 9
locks on shared resources 7
logging 88, 100
see stable recording 8
loosely-coupled thread 93
machine failure at RM 17
managing global transactions
HTL 98
STDL 98
mapping 85
mapping of XID 20
RM requirement 66
mapping to CRM
HTL 98
STDL 98
matching heuristic decisions 9
MAXINFOSIZE 21
method of referencing transaction 5
migration 88, 94
migration, declaring RM support 21
mixed-heuristic 89
model 1, 75
benefits 76
definition 79-80
functional 4, 82
instance 79
modes of xa_calls 17
modifying shared resource 5
monitoring not addressed 76
multiple
access to RMs 66
associations 14
state transitions 57
threads using RM 14
name of RM 21, 67
name registration 67
naming conventions 19
naming not addressed 76
native
interface 65, 68
statements 68
native interface 78, 85
constraints 85
negative response to pre-commit 8
non-ANSI C
see Common Usage C 19
non-blocking mode 17
TM option 69
non-global transaction
definition 99
with global 99
non-standard native open 13
notation, in state tables 58
notification of heuristic decisions 9
null string permitted 67
null XID 20
object modules 65
octet string 66
one-phase commit 9
failure implications 9
TM option 69
open, non-standard native 13
operations known within RM 6
optimisation 16, 88
of commitment protocol 8
read-only 67
RM option 67

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

optional features....65
orderly routine....67
OSI CCR
  compatibility with....9
  use in XID....19
OSI CCR standards....76, 87
  atomic action identifier....87
OSI TP....8, 76
  heuristic completion....9
  heuristics....89
  recovery....95
OSI TP standards....77-78, 80, 84, 86, 91
  recovery....90
OSI TP-CRM interface....86
overview of interface....11
parametric timeout....89
performance....100
permanence of changes....16
phase 1....88
phase 2....88
phases of commitment protocol....8
platforms....76
pointer to XID....20
pointers to RM entry points....21
portability....65, 75-76
  enhancing....13
  maximising....68
pre-commit
  negative response to....8
pre-committed transactions....9
prepare (to commit)....88
prepare to commit....8, 16
prepared transactions
  list of....17
preserving XID....20
presumed rollback....8-9, 89
print server, implemented as RM....5
procedures
  HTL....98
  STDL....98
process....7
  relationship with component....100
products
  existing....102
protocol....2, 75, 88, 94
  commitment....8
  error....57
  optimisation....88
  optimisations....8
  RM requirement....66
  state table....61
public information
  RM requirement....67
  TM requirement....69
publication of XID, TM option....69
rarely-used RMs....15
RDBMS....68
read-only optimisation....88
read-only response....8
recompilation, avoiding....19, 65
recovery....6, 9, 90, 95
  from failure....17
  list....17
  local by TM....9
  RM requirement....66
  TM....83
registration
  of product name....67
  of RM....15
registration of RM....15
registration, dynamic....15
rejoining transaction....14
releasing resources....16
report on completion....88
reporting
  failure to prepare to commit....8
  transaction information....12
request for work....88, 94
requirements for implementors....65
resource....1, 75
  access to....1, 75
  database....1, 75
  file access system....1, 75
  heuristic decision....89, 95
  manager....1, 75
  restoring consistency....90, 95
  RM....78
resource manager....5
  ACID properties responsibility....77
  component....83
  distributed....76
  dynamic registration....88
  heuristic decision....89
  informed of start of transaction....88, 94
  interface to AP....85
  interface to TM....85
  internal transaction....99
  native interface....78
  shared resource....78, 88
  unit of work....99
resource manager native interface
  definition....78

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

resource manager resource
definition 78
resources
AP access to 5
releasing 16
system 7
result
mixed-heuristic 89
result codes 24
return codes 24
RM 1, 75
access to stable storage 9
ACID properties responsibility 77
changing sets of 21
component 83
concurrent use of 14
distributed 76
dynamic control of participation 12
dynamic registration 88
entry points 21
failure, state transition 58
heuristic decision 89
informed of start of transaction 88, 94
initialisation state table 58
machine failure 17
name 67
name of 21
native interface 78
opening and closing 13
option 67-68
re-registerings 15
registration of 15
requirements 66
return 69
role in recovery 90
sequencing use of 14
serialising use of 14
shared resource 78, 88
start-up actions 13
switch 21, 69
unilateral action 17
unit of work 99
updating shared resources 9
RM access
HTL 98
STDL 98
RM native interface
definition 78
RM resource
definition 78
RM-AP interface 85
RM-internal transaction
definition 99
RM-TM interface 85
RMNAMESZ 21
RMs
work done across 5
rollback 78, 95
explicit 89
implicit 89
presumed 88-89, 94
rollback only 15
RM option 17
ROLLBACK WORK statement in SQL 68
rolling back transactions 5, 8, 16
routine, unused 67
security not addressed 76
sequence
of calls 57
of transaction 5
of xa_ routines 12
sequencing access 66
serialisability, guaranteeing 8
serialising access 66
server, implemented as RM 5
service interfaces
TM requirements 69
services, index 12
shared resource
restoring consistency 90, 95
RM 78, 83
update 88
shared resources 7
changes to 16
modifying 5
no update to 8
permanence of changes to 5
RM management of 5
unlocking 9
simplification of design 76
simultaneous updates, across RMs 6
single asynchronous operation 63
skeleton routines 65
SNMP 100
specification
CPI-C Version 2 interface 83, 86, 97
TX interface 85, 91
TxRPC interface 83, 86, 96
XA interface 84-85, 88
XA+ interface 83, 86, 94
XAP-TP interface 86
XATMI interface 83, 86, 96

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SQL
COMMIT 99
interface 78, 85
stable recording 8
stable storage, access 9
standards
international 76
OSI CCR 76, 87
OSI TP 76-78, 80, 84, 86, 90-91
start-up actions in RM 13
state table 57
closing RM 58
failure return 58
general entry 58
initial state in 57
opening RM 58
RM requirement to enforce 66
specific entry 58
status of work done 5
STDL
AP to AP communication 98
CRM independence 98
language 80, 98
language-syntax level 98
managing global transactions 98
mapping to CRM 98
procedures 98
RM access 98
transactional 98
TxRPC protocol mapping 98
subordinate AP 83
subordinate TM 83
superior AP 83
superior name (OSI CCR) 20
superior TM 83
support for recovery 66
suspending association with transaction 14
switch name 67
switch, RM 21
synchronous calling mode 17
syntax, in state tables 58
system component, failure of 5
system management
CMIP 100
distributed 100
SNMP 100
system-level interface 1, 75
system-specific procedure 13, 65
table, state 57
template names of xa_ routines 12
terminated thread 17
termination string 13, 67
locating 69
testing for completion 17
text strings
configuration control with 65
RM requirement to publish 67
thread 80
association with transactions 14
loosely-coupled 93
migration 88, 94
same across calls 80
supervising completion 16
tightly-coupled 93
transaction association 88, 94
transaction branch 88, 94
thread of control 7
association state table 59
manner of identifying 66
same across calls 7
state transition 57
termination 17
tightly-coupled thread 93
timeout 89
TM 1, 6, 75, 83
access to stable storage 9
ACID properties coordination 77
API 85
atomicity 83
delegating responsibility to 65
linking assumptions 69
options 69
performing local recovery 9
prefix 19
recovery 83
requirements 69
role in recovery 90
TM domain 91
definition 79
sharing resources 91
TM-AP interface 85
TM-CRM interface 86
TM-RM interface 85
TMASYNC
flag definition 22
in state table 63
in xa_close( ) 32-33
in xa_commit( ) 34-36
in xa_complete( ) 37
in xa_end( ) 39-40
in xa_forget( ) 41-42
in xa_open( ) 43-44

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

in xa_prepare( ) 45-46
in xa_rollback( ) 50-52
in xa_start( ) 54-55
TMENDRSCAN
flag definition 22
in xa_recover( ) 48
TMER_ prefix 19
TMER_INVAL
in ax_reg( ) 29
in ax_unreg( ) 31
return code 24
TMER_PROTO
in ax_reg( ) 29
in ax_unreg( ) 31
return code 24
TMER_TMERR
in ax_reg( ) 29
in ax_unreg( ) 31
return code 24
TMFAIL
flag definition 22
in state table 59-60
in xa_end( ) 38-39
TMJOIN
flag definition 22
in xa_start( ) 53, 55
TMMIGRATE
flag definition 22
in ax_reg( ) 29
in xa_end( ) 38-39
in xa_start( ) 54
TMMULTIPLE
flag definition 22
in xa_complete( ) 37
TMNOFLAGS 22
flag definition 22
in ax_reg( ) 29
in ax_unreg( ) 31
in xa_close( ) 32
in xa_commit( ) 34
in xa_complete( ) 37
in xa_forget( ) 41
in xa_open( ) 43
in xa_prepare( ) 45
in xa_recover( ) 48
in xa_rollback( ) 50
in xa_start( ) 54
TMNOMIGRATE
flag definition 22
in xa_end( ) 38-39
TMNOWAIT
flag definition 22
in state table 63
in xa_commit( ) 34-35
in xa_complete( ) 37
in xa_start( ) 54
TMONEPHASE
flag definition 22
in xa_commit( ) 34-35
TMREGISTER
flag definition 22
in ax_reg( ) 28-29
in ax_unreg( ) 31
in xa_end( ) 38
in xa_start( ) 53
TMRESUME
flag definition 22
in state table 59
in xa_start( ) 53-55
TMSTARTRSCAN
flag definition 22
in xa_recover( ) 48-49
TMSUCCESS
flag definition 22
in state table 59-60
in xa_commit( ) 34
in xa_end( ) 38-39
in xa_prepare( ) 45
TMSUSPEND
flag definition 22
in state table 59-60
in xa_end( ) 38-39
TMUSEASYNC
flag definition 22
in xa_close( ) 32
in xa_commit( ) 35
in xa_end( ) 40
in xa_forget( ) 41
in xa_open( ) 43
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 55
TM_ prefix 19
TM_JOIN
in ax_reg( ) 29
return code 24
TM_OK
in ax_reg( ) 29
in ax_unreg( ) 31
return code 24

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TM_RESUME
in ax_reg( ) 29
in state table 60
return code 24
TP language
HTL 75, 83, 98
STDL 80, 98
TP monitor 99, 101
tracking transactions 66
transaction
ACID properties 89
actions 1, 75
associating threads 14
association 88, 94
association state table 59
atomicity 88
authorisation to forget 9
blocking 88, 94
boundary 83
branch 88, 94
branch identifier 6
commit 78, 88, 94
commit coordination 93
commit decision 83
commit protocol 78
commitment and recovery 8
commitment protocol 8
committing 5, 16
completion 1, 16, 75, 78, 83
context 7, 15
controlling RM participation 12
defining boundaries 1, 75
definition 77
definition of 5
demarcation 83, 85
dissociating from 14
dissociating threads 14
distributed 78
ending association 14
ending involvement in 8
explicit rollback 89
failure 1, 75, 90, 95
forgetting 16, 88
global 1, 75, 78, 83-84, 99
global demarcation 91
global tree structure 91-92
global, definition of 6
heuristic completion 89, 95
heuristic outcome 78
HTL 98
identifier (XID) 87
identifier assigning 1, 75
implicit rollback 89
incomplete state 14
independent completion 9
inform RMs of start 88, 94
information reporting 12
initiation 88, 94
joining 14
knowledge 88
location-independence of work 5
manager 1, 6, 75
method of referencing 5
migration 88, 94
non-global 99
pre-committed 9
presumed rollback 89
properties 77
recording data 88
recovery 1, 75, 90, 95
resuming association 14
RM native interface 78
RM resource 78
RM-internal 6, 99
RMs must recognise 66
rollback 78, 88-89, 95
rollback-only 15
rolling back 5, 8
start of association 14
state table 61
STDL 98
suspending association 14
thread 88, 94
two-phase commit 78, 88
work outside 15
transaction completion
definition 78
transaction ID
component of XID 19
transaction manager
ACID properties coordination 77
API 85
atomicity 83
interface to AP 85
interface to CRM 86
interface to RM 85
recovery 83
role in recovery 90
transaction manager domain 91
definition 79
sharing resources 91
transaction processing monitor 99, 101

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

transition 57
two-phase commit 8
  definition 78
TX interface 80, 85, 91
TxRPC interface 80, 83-84, 86, 96
TxRPC protocol mapping
  STDL 98
undoing changes 16
undoing work 5
unilateral RM action 17
unimplemented routines 67
uniqueness of XID 19, 69
unit of work 5, 99
unused routines 67
update shared resource 88
update, lack of 8
user interface not addressed 76
vendor
  options for RMs 67
  requirements for TMs 69
vendor requirements 65
vendor-specific procedure 65
version word 21
veto 8
  reporting by xa_start 14
work done
  across RMs 5
  status of 5
work outside transaction 15
X/Open consensus 76
X/Open publications 75
X/Open specification
  CPI-C Version 2 interface 83, 86, 97
  TX interface 85, 91
  TxRPC interface 83, 86, 96
  XA interface 84-85, 88
  XA+ interface 83, 86, 94
  XAP-TP interface 86
  XATMI interface 83, 86, 96
X/Open-compliant interface 5
XA interface 1, 80, 84-85, 88
  alternative 85
  visibility 85
XA+ interface 80, 83-84, 86, 94
xa.h header 19
XAER_ prefix 19
XAER_ASYNC
  in xa_close( ) 32
  in xa_commit( ) 35
  in xa_end( ) 40
  in xa_forget( ) 41
  in xa_prepare( ) 46
  in xa_rollback( ) 51
  in xa_start( ) 55
  return code 25
XAER_DUPID
  in xa_start( ) 55
  return code 25
XAER_INVAL
  in xa_close( ) 33
  in xa_commit( ) 36
  in xa_complete( ) 37
  in xa_end( ) 40
  in xa_forget( ) 41
  in xa_open( ) 44
  in xa_prepare( ) 46
  in xa_recover( ) 49
  in xa_rollback( ) 52
  in xa_start( ) 55
  return code 25
XAER_NOTA
  in xa_commit( ) 36
  in xa_end( ) 40
  in xa_forget( ) 41
  in xa_prepare( ) 46
  in xa_rollback( ) 52
  in xa_start( ) 55
  return code 25
XAER_OUTSIDE
  in xa_start( ) 55
  return code 25
XAER_PROTO
  in state table 57
  in xa_close( ) 33
  in xa_commit( ) 36
  in xa_complete( ) 37
  in xa_end( ) 40
  in xa_forget( ) 41
  in xa_open( ) 44
  in xa_prepare( ) 46
  in xa_recover( ) 49
  in xa_rollback( ) 52
  in xa_start( ) 55
  return code 25
XAER_RMERR
  in state table 62
  in xa_close( ) 32
  in xa_commit( ) 36
  in xa_end( ) 40
  in xa_forget( ) 41
  in xa_open( ) 44
  in xa_prepare( ) 46

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

in xa_recover( ) 49
in xa_rollback( ) 51
in xa_start( ) 55
return code 25
XAER_RMFAIL
in state table 59-60, 62
in xa_commit( ) 36
in xa_end( ) 40
in xa_forget( ) 41
in xa_prepare( ) 46
in xa_recover( ) 49
in xa_rollback( ) 51
in xa_start( ) 55
return code 25
XAP-TP interface 80, 84, 86
XATMI interface 80, 83-84, 86, 96
XA_ prefix 19
xa_ routines 12
names of 12
order of use 12
sequence of 12
xa_close( ) 12, 32
redundant use 58
state table 58
xa_commit( ) 12, 16, 34
xa_complete( ) 12, 17, 37
xa_end( ) 12, 14-15, 38
RMs applied to 15
xa_forget( ) 12, 41
XA_HEUR*
in state table 62
XA_HEURCOM
in xa_commit( ) 35
in xa_rollback( ) 51
return code 24
XA_HEURHAZ
in xa_rollback( ) 50
return code 24
XA_HEURMIX
in xa_commit( ) 35
in xa_rollback( ) 51
return code 24
XA_HEURRB
in xa_commit( ) 35
in xa_rollback( ) 51
return code 24
XA_NOMIGRATE
in xa_end( ) 39
return code 24
XA_OK
in state table 62
in xa_close( ) 32
in xa_commit( ) 35
in xa_complete( ) 37
in xa_end( ) 39
in xa_forget( ) 41
in xa_open( ) 43
in xa_prepare( ) 45
in xa_rollback( ) 51
in xa_start( ) 54
return code 24
xa_open( ) 12, 43
redundant use 58
RM parameters passed in 13
state table 58
xa_prepare( ) 12, 16, 45
required 66
RMs applied to 15
XA_RB*
in state table 59, 62
in xa_commit( ) 35
in xa_end( ) 39
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 54
XA_RBBASE
return code 24
XA_RBCOMMFAIL
in xa_commit( ) 35
in xa_end( ) 39
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 54
return code 24
XA_RBDEADLOCK
in xa_commit( ) 35
in xa_end( ) 39
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 55
return code 24
XA_RBEND
return code 24
XA_RBINTEGRITY
in xa_commit( ) 35
in xa_end( ) 39
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 55
return code 24
XA_RBOTHER
in xa_commit( ) 35

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

in xa_end( ) 39
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 55
return code 24
XA_RBPROTO
in xa_commit( ) 35
in xa_end( ) 39
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 55
return code 24
XA_RBROLLBACK
in xa_commit( ) 35
in xa_end( ) 39
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 54
return code 24
XA_RBTIMEOUT
in xa_commit( ) 35
in xa_end( ) 39
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 55
return code 24
XA_RBTRANSIENT
in xa_commit( ) 35
in xa_end( ) 40
in xa_prepare( ) 46
in xa_rollback( ) 51
in xa_start( ) 55
return code 24
XA_RDONLY
in state table 62
in xa_prepare( ) 45
return code 24
xa_recover( ) 12, 17, 48
XA_RETRY
in state table 62-63
in xa_commit( ) 35
in xa_complete( ) 37
in xa_start( ) 54
return code 24
xa_rollback( ) 12, 16, 50
xa_start( ) 12, 14-15, 53
not used in dynamic registration 15
primary use 14
RMs applied to 15
to resume 14
xa_switch_t 21
XID 6, 87
global uniqueness 87
local copy 20
mapping to local transaction 20
null 20
structure and byte alignment 19
structure definition 19
TMs required to manage 69
uniqueness 66

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

© ISO/IEC

ICS 35.100

Descriptors: data processing, information interchange, network interconnection, open systems interconnection, computer components, computer interfaces, application program interfaces.

Price based on 111 pages

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale