Technical Standard

Distributed Transaction Processing: The TX (Transaction Demarcation) Specification

THE Open GROUP
[This page intentionally left blank]
X/Open CAE Specification

Distributed Transaction Processing:
The TX (Transaction Demarcation) Specification

X/Open Company Ltd.
X/Open CAE Specification
Distributed Transaction Processing: The TX (Transaction Demarcation) Specification
X/Open Document Number: C504

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

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

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

or by Electronic Mail to:

XoSpecs@xopen.co.uk
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>Introduction</td>
</tr>
<tr>
<td>1.1</td>
<td>X/Open DTP Model</td>
</tr>
<tr>
<td>1.2</td>
<td>X/Open TX (Transaction Demarcation) Interface</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Model and Definitions</td>
</tr>
<tr>
<td>2.1</td>
<td>X/Open DTP Model</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Functional Components</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Interfaces between Functional Components</td>
</tr>
<tr>
<td>2.2</td>
<td>Definitions</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Transaction</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Transaction Properties</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Distributed Transaction Processing</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Global Transactions</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Thread of Control</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>C-language Interface Overview</td>
</tr>
<tr>
<td>3.1</td>
<td>Index to Services in the TX Interface</td>
</tr>
<tr>
<td>3.2</td>
<td>Opening and Closing Resource Managers</td>
</tr>
<tr>
<td>3.3</td>
<td>Beginning and Completing Global Transactions</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Heuristic Completion</td>
</tr>
<tr>
<td>3.4</td>
<td>Setting the Commit Return Point</td>
</tr>
<tr>
<td>3.5</td>
<td>Chained and Unchained Transactions</td>
</tr>
<tr>
<td>3.6</td>
<td>Transaction Timeout</td>
</tr>
<tr>
<td>3.7</td>
<td>Information on the Global Transaction</td>
</tr>
<tr>
<td>3.8</td>
<td>Transaction Characteristics</td>
</tr>
<tr>
<td>3.8.1</td>
<td>The commit_return Characteristic</td>
</tr>
<tr>
<td>3.8.2</td>
<td>The transaction_control Characteristic</td>
</tr>
<tr>
<td>3.8.3</td>
<td>The transaction_timeout Characteristic</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>The <code>&lt;tx.h&gt;</code> Header</td>
</tr>
<tr>
<td>4.1</td>
<td>Naming Conventions</td>
</tr>
<tr>
<td>4.2</td>
<td>Transaction Information</td>
</tr>
<tr>
<td>4.3</td>
<td>Return Codes</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>C Reference Manual Pages</td>
</tr>
<tr>
<td>(tx_begin()</td>
<td>20</td>
</tr>
<tr>
<td>tx_close()</td>
<td>22</td>
</tr>
<tr>
<td>tx_commit()</td>
<td>23</td>
</tr>
<tr>
<td>tx_info()</td>
<td>25</td>
</tr>
<tr>
<td>tx_open()</td>
<td>26</td>
</tr>
<tr>
<td>tx_rollback()</td>
<td>27</td>
</tr>
<tr>
<td>tx_set_commit_return()</td>
<td>29</td>
</tr>
</tbody>
</table>
X/Open

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

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

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

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

X/Open Technical Publications

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

There are two types of X/Open specification:

- **CAE Specifications**

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

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

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

These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

Preliminary specifications are analogous to the trial-use standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- **Guides**

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- **Technical Studies**

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

- **Snapshots**

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

Versions and Issues of Specifications

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

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

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

Corrigenda

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

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

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

request corrigenda; topic index

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

This Document

This document is a CAE Specification (see above). It defines the TX (Transaction Demarcation) interface, which is the interface between an application program and a transaction manager.

The structure of the specification is as follows:

• Chapter 1 is an introduction.
• Chapter 2 provides fundamental definitions for the remainder of the document.
• Chapter 3 is an overview of the TX interface.
• Chapter 4 discusses the data structures that are part of the TX interface.
• Chapter 5 contains C reference manual pages for each routine in the TX interface.
• Chapter 6 contains COBOL reference manual pages for each routine in the TX interface.
• Chapter 7 contains the state table that describes the legal sequences in which calls to the TX interface can be made.
• Chapter 8 summarises the implications of this specification on implementors.
• Appendix A contains a complete C header file that is required by the TX interface.
• Appendix B describes how the TX interface maps to the X/Open XA interface.

There is an index at the end.

Intended Audience

This document is intended for application programmers who wish to write portable programs that use global transactions. It assumes familiarity with the X/Open documents Distributed Transaction Processing Reference Model and Distributed Transaction Processing: The XA Specification.
Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for filenames, keywords, type names, data structures and their members.

- **Italic** strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - variable names, for example, substitutable argument prototypes and environment variables
  - C-language functions; these are shown as follows: `name()`

- Normal font is used for the names of constants and literals. COBOL function names are also shown in normal font.

- The notation `<file.h>` indicates a C-language header file.

- Names surrounded by braces, for example, `{ARG_MAX}`, represent symbolic limits or configuration values, which may be declared in appropriate C-language header files by means of the C `#define` construct.

- The notation `[ABCD]` is used to identify a coded return value in C, or the value set in COBOL.

- Syntax and code examples are shown in **fixed width** font.

- Variables within syntax statements are shown in **italic fixed width** font.

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

- No underlining is used with mandatory elements.

- No options are shown; for other valid formats see the X/Open **COBOL Language** specification.

- Substitutable names are shown in italics.
Trade Marks

X/Open® is a registered trade mark, and the “X” device is a trade mark, of X/Open Company Limited.
Referenced Documents

COBOL

CPI-C, Version 2

DTP

ISO C

OSI TP
ISO/IEC 10026-1: 1992, Information Technology — Open Systems Interconnection — Distributed Transaction Processing, Parts 1 to 3:
  Part 1: OSI TP Model
  Part 2: OSI TP Service

SQL

TxRPC

XA

XA+

XAP-TP

XATMI

XDCS
This chapter provides an outline of the X/Open Distributed Transaction Processing (DTP) model and gives an overview of the X/Open TX (Transaction Demarcation) Interface.

1.1 X/Open DTP Model

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

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

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

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

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

Chapter 2 defines each component in more detail and illustrates the flow of control.
1.2 X/Open TX (Transaction Demarcation) Interface

This document specifies the TX (Transaction Demarcation) interface; the application programming interface (API) by which the AP calls the TM to demarcate global transactions and direct their completion. X/Open is developing other interfaces for distributed transaction processing; see the referenced DTP guide for an overview of the complete set of interfaces specified for DTP.

Relevant definitions and other important concepts are discussed in Chapter 2. Chapter 3 is an overview of the TX interface, describing the situations in which each of the services is used. Chapter 4 discusses the data structures that are part of the TX interface. C Reference manual pages for each routine in the TX interface are presented in Chapter 5 and COBOL manual pages are presented in Chapter 6. Chapter 7 shows the legal sequences in which routines in the TX interface may be called. Chapter 8 summarises the implications of this specification on the implementors of APs and TMs; it also identifies features that are optional. Appendix A presents the contents of the <tx.h> header file in both ISO C and Common Usage C. Appendix B discusses the relationship of this interface to the interface published in the XA specification. This appendix describes how the TX services map to the XA services, and how the XA return codes map to the TX return codes.
This chapter discusses the TX interface in general terms and provides necessary background material for the rest of the specification. The chapter shows the relationship of the interface to the X/Open DTP model. The chapter also states the design assumptions that the interface uses and shows how the interface addresses common DTP concepts.

### 2.1 X/Open DTP Model

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

![Diagram of functional components and interfaces](image)

**Figure 2-1** Functional Components and Interfaces

Descriptions of the functional components shown can be found in Section 2.1.1 on page 4. The numbers in brackets in the above figure represent the different X/Open interfaces that are used in the model. They are described in Section 2.1.2 on page 5.

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

---

Distributed Transaction Processing: The TX (Transaction Demarcation) Specification 3
2.1.1 Functional Components

Application Program (AP)

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

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

Transaction Manager (TM)

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

Resource Manager (RM)

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

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

Communication Resource Manager (CRM)

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

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

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

There are six interfaces between software components in the X/Open DTP model. The numbers correspond to the numbers in Figure 2-1 on page 3.

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

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

For details of the AP-TM interface, see this specification.

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

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

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

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

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

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

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

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

X/Open interfaces, such as the CRM APIs listed above, provide application portability. 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.
2.2 Definitions

For additional definitions see the referenced DTP guide.

2.2.1 Transaction

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

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

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

2.2.2 Transaction Properties

Transactions typically exhibit the following properties:

Atomicity 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.

2.2.3 Distributed Transaction Processing

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

• The system must have a way to refer to a transaction that encompasses all work done anywhere in the system.

• The decision to commit or roll back a transaction must consider the status of work done anywhere on behalf of the transaction. The decision must have uniform effect throughout the DTP system.

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

Every RM in the DTP environment must support transactions as described in Section 2.2.1 on page 7. Many RMs already structure their work into recoverable units.

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

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

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. The context may include locks on shared resources and open files. 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 transactions. The way the RM knows that a given work request pertains to a given transaction 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.
This chapter gives an overview of the TX interface. TX is the interface between the AP and the TM in an X/Open DTP system. Chapter 5 contains reference manual pages for each routine in alphabetical order. These pages contain a C-language definition of each function. The COBOL manual pages for the equivalent routines are presented in Chapter 6. This chapter describes the C-language interface.

The TX routines are supplied by TMs operating in the DTP environment and are called by APs. APs demarcate global transactions via the TX interface and perform recoverable operations via RMs’ native interfaces.

Figure 3-1 The TX (Transaction Demarcation) Interface
3.1 Index to Services in the TX Interface

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_begin</td>
<td>Begin a global transaction.</td>
<td>Section 3.3 on page 11.</td>
</tr>
<tr>
<td>tx_close</td>
<td>Close a set of resource managers.</td>
<td>Section 3.2.</td>
</tr>
<tr>
<td>tx_commit</td>
<td>Commit a global transaction.</td>
<td>Section 3.3 on page 11.</td>
</tr>
<tr>
<td>tx_info</td>
<td>Return global transaction information.</td>
<td>Section 3.7 on page 12.</td>
</tr>
<tr>
<td>tx_open</td>
<td>Open a set of resource managers.</td>
<td>Section 3.2.</td>
</tr>
<tr>
<td>tx_rollback</td>
<td>Roll back a global transaction.</td>
<td>Section 3.3 on page 11.</td>
</tr>
<tr>
<td>tx_set_commit_return</td>
<td>Set commit_return characteristic.</td>
<td>Section 3.4 on page 11.</td>
</tr>
<tr>
<td>tx_set_transaction_control</td>
<td>Set transaction_control characteristic.</td>
<td>Section 3.5 on page 12.</td>
</tr>
<tr>
<td>tx_set_transaction_timeout</td>
<td>Set transaction_timeout characteristic.</td>
<td>Section 3.6 on page 12.</td>
</tr>
</tbody>
</table>

Table 3-1 C-Language TX Functions

3.2 Opening and Closing Resource Managers

An AP must call tx_open() to open all RMs linked with the AP. If tx_open() returns with an error, all RMs are closed. If tx_open() completes successfully, some or all of the RMs are open. RMs that are not open return RM-specific errors when accessed by the AP. Only upon successful completion of the tx_open() call can an AP execute global transactions.

The AP calls tx_close() to close all RMs linked with the AP. For tx_close() to return success, the AP cannot be part of an active global transaction when tx_close() is called.

The TM opens and closes all RMs that are linked to the application (see the XA specification). Since RMs differ in their initialisation and termination semantics, the RM vendor must publish specific information the TM needs to open and close a particular RM.

Allowing the TM to open and close RMs keeps the AP from having to include RM-specific calls that may hinder portability.
3.3 **Beginning and Completing Global Transactions**

An AP calls `tx_begin()` to mark the beginning of a global transaction. This act places the calling AP in *transaction mode*. The function fails if the caller is already in transaction mode, without affecting the transaction active in the thread of control.

An AP calls `tx_commit()` to direct the TM to commit its global transaction. The TM coordinates transaction commitment with all RMs involved in the transaction.

An AP calls `tx rollback()` to direct the TM to roll back a global transaction. The TM directs RMs to undo any changes to recoverable resources that they made within the given global transaction.

A call to either `tx_commit()` or `tx rollback()` begins a new global transaction when the previous one is completed, if the AP has selected chained mode. (See Section 3.5 on page 12.)

Many RMs let APs request work outside any global transaction without coordination by an X/Open TM. This can happen when the AP is not in transaction mode and does work with one or more RMs.

3.3.1 **Heuristic Completion**

Under certain unusual failure conditions, an RM may unilaterally commit or roll back changes to recoverable resources that it made within a global transaction. If this decision does not match the commit decision the TM makes, a return code on `tx_commit()` or `tx rollback()` notifies the AP of a mixed heuristic outcome. This means that some parts of the transaction have been committed and some have been rolled back. In some cases, a TM may not be able to determine whether an RM’s decision matched the commit decision made by the TM. In such a situation, a separate return code on `tx_commit()` or `tx rollback()` notifies the AP that some RMs may have made a heuristic decision (this is known as a heuristic hazard condition). Only heuristic decisions that are mixed or hazard are reported to the AP as heuristic decisions.

3.4 **Setting the Commit Return Point**

The function `tx_commit()` normally returns when the two-phase commit procedure is completed. An AP can choose to return from `tx_commit()` at the point when the decision to commit is logged but prior to completing the second phase. This is known as an early return. An application designer might employ early return to reduce response time to the user. However, if a transaction has a heuristic outcome (see Section 3.3.1), a call to `tx_commit()` that returns early does not indicate this outcome.

The AP calls `tx_set_commit_return()` to enable or disable early return from `tx_commit()`. This call can be issued whether or not the AP is in transaction mode. The setting remains in effect until overridden by another call to `tx_set_commit_return()`.
3.5 Chained and Unchained Transactions

The TX interface supports chained and unchained modes of transaction execution. By default, an AP executes in the unchained mode; when an active global transaction is completed, a new transaction does not begin until the AP calls `tx_begin()`.

In the chained mode, a new transaction starts in the AP thread of control implicitly when the current transaction completes. That is, when the AP calls `tx_commit()` or `tx_rollback()`, the TM coordinates completion of the current transaction and initiates a new transaction in the calling thread of control before returning control to the AP. (Certain failure conditions may prevent a new transaction from starting.)

The AP enables or disables the chained mode by calling `tx_set_transaction_control()`. Transitions between the chained and unchained mode affect the behaviour of the next `tx_commit()` or `tx_rollback()` call. The call to `tx_set_transaction_control()` does not put the AP into or take it out of a global transaction.

Since `tx_close()` cannot be called when an AP is in a global transaction, the AP executing in chained mode must switch to unchained mode and complete the current transaction before calling `tx_close()`.

3.6 Transaction Timeout

To control the amount of resource spent executing a transaction, an AP may use `tx_set_transaction_timeout()` to set the `transaction_timeout` value. This value specifies the time period (in seconds) in which the transaction must complete before it becomes susceptible to transaction timeout. A value of 0 means that the timeout feature is disabled. The initial setting for each AP is 0.

An AP may call `tx_set_transaction_timeout()` at any time, but the call affects only transactions the AP begins subsequently. If an AP calls `tx_set_transaction_timeout()` in transaction mode, the new timeout value does not affect the current transaction.

The timeout interval begins and is reset whenever the AP calls `tx_begin()`. It is also reset by `tx_commit()` or `tx_rollback()` in the chained mode when these functions start a new transaction. If the AP does not call either `tx_commit()` or `tx_rollback()` before the interval expires, the TM declares a timeout whenever it is possible to do so (under certain circumstances the TM may not be able to interrupt an operation in progress). A transaction for which the TM has declared a timeout is rolled back.

Transaction state information returned by `tx_info()` indicates that a transaction has timed out and has been marked rollback-only. To complete such a transaction, the AP should call `tx_rollback()`. If an AP issues a `tx_commit()` for a transaction that has been marked rollback only, the TM rolls back the transaction.

3.7 Information on the Global Transaction

An AP may call `tx_info()` to obtain the current values of the transaction characteristics affecting the AP thread of control (see Section 3.8 on page 13), to determine whether the AP is executing in transaction mode, and to obtain transaction state information.
3.8 Transaction Characteristics

The state of an application thread of control includes several characteristics. The AP specifies these by calling \textit{tx\_set\_\_\textit{characteristic}(\()\) functions. When the AP sets the value of a characteristic, it remains in effect until the AP specifies a different value. When the AP obtains the value of a characteristic, it does not change the value.

The following table is a complete list of the set of characteristics that pertain to each global transaction:

<table>
<thead>
<tr>
<th>Characteristic Name</th>
<th>Type</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit_return</td>
<td>COMMIT_RETURN</td>
<td>Section 3.8.1.</td>
</tr>
<tr>
<td>transaction_control</td>
<td>TRANSACTION_CONTROL</td>
<td>Section 3.8.2.</td>
</tr>
<tr>
<td>transaction_timeout</td>
<td>TRANSACTION_TIMEOUT</td>
<td>Section 3.8.3.</td>
</tr>
</tbody>
</table>

Table 3-2 Table of Transaction Characteristics

3.8.1 The commit\_return Characteristic

The \textit{commit\_return} characteristic determines the stage in the commitment protocol at which the \textit{tx\_commit()} call returns to the AP. The definition of this characteristic is as follows:

```c
typedef long COMMIT\_RETURN;
#define TX\_COMMIT\_COMPLETED 0
#define TX\_COMMIT\_DECISION\_LOGGED 1
```

For more information, see Section 3.4 on page 11, \textit{tx\_commit()} on page 23 or \textit{tx\_set\_commit\_return()} on page 29.

3.8.2 The transaction\_control Characteristic

The \textit{transaction\_control} characteristic determines whether the completion of one transaction automatically begins a new transaction (called \textit{chained} mode). The definition of this characteristic is as follows:

```c
typedef long TRANSACTION\_CONTROL;
#define TX\_UNCHAINED 0
#define TX\_CHAINED 1
```

For more information, see Section 3.5 on page 12, or \textit{tx\_set\_transaction\_control()} on page 31.

3.8.3 The transaction\_timeout Characteristic

The \textit{transaction\_timeout} value specifies the time period in which the transaction must complete before becoming susceptible to transaction timeout. The interval is expressed as a number of seconds. The definition of this characteristic is as follows:

```c
typedef long TRANSACTION\_TIMEOUT;
```

For more information, see Section 3.6 on page 12, or \textit{tx\_set\_transaction\_timeout()} on page 32.
This chapter specifies C-language structure definitions, argument values, and return codes to which conforming products must adhere. These, plus the function prototypes for the interface routines defined in the next chapter, are the minimum required contents of the C-language header file `<tx.h>`.

Appendix A contains a `<tx.h>` header file with `#define` statements suitable for ISO C and Common Usage C implementations. This chapter contains excerpts from the ISO C code in `<tx.h>`. The synopses in Chapter 5 also use ISO C.

### 4.1 Naming Conventions

The C interface to TX uses certain naming conventions to name its functions, flags, and return codes. All names that appear in `<tx.h>` are part of the TX name space. This section describes the TX naming conventions.

- The names of all TX routines begin with `tx_` (for example, `tx_begin()`).
- The names of return codes and of types and constants used in arguments to TX routines begin with `TX_`.

### 4.2 Transaction Information

The `tx_info_t` structure is used to return information about the thread state, including the state of all characteristics (see Section 3.8 on page 13), the thread’s association, if any, to a global transaction, and transaction state information.

```c
struct tx_info_t {
    TXID xid;
    COMMIT_RETURN when_return;
    TRANSACTION_CONTROL transaction_control;
    TRANSACTION_TIMEOUT transaction_timeout;
    TRANSACTION_STATE transaction_state;
};

typedef struct tx_info_t TXINFO;
```

The `<tx.h>` header defines a public structure called an XID to identify a transaction branch. The XID structure is specified in the C code below in `struct xid_t`. The AP may call `tx_info()` to obtain XID information to identify in which global transaction it is located; for example, to assist in auditing and debugging. If the AP is not in any global transaction, the TM sets the xid field to the null XID. The XID contains a format identifier, two length fields, and a data field. The data field comprises two contiguous components: a global transaction identifier (gtrid) and a branch qualifier (bqual).

APs may use XIDs for administrative purposes such as auditing and logging, but cannot use them to affect the TM’s coordination of the global transaction. (None of the TX calls accept an XID as an input parameter).

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 contents of the unused bytes in `data` are undefined.

An important attribute of the `XID` is global uniqueness, based on the exact order of the bits in the `data` element of the `XID` for the lengths specified. The AP should treat each component of `data` as an arbitrary collection of octets because, for instance, a component may contain binary data as well as printable text.

```c
#define XIDDATASIZE 128 /* size in bytes */
struct xid_t {
    long formatID; /* format identifier */
    long gtrid_length; /* value not to exceed 64 */
    long bqual_length; /* value not to exceed 64 */
    char data[XIDDATASIZE]; /* may contain binary data */
};
typedef struct xid_t XID;
/*
 * A value of -1 in formatID means that the XID is null.
 */
```

The `TRANSACTION_STATE` element is specified in the C code below. The AP may call `tx_info()` to obtain information regarding the state of the transaction it is in; that is, to determine whether the transaction is active, has timed-out (and been marked rollback-only), or has been marked rollback-only (for a reason other than transaction timeout).

```c
typedef long TRANSACTION_STATE;
#define TX_ACTIVE 0
#define TX_TIMEOUT_ROLLBACK_ONLY 1
#define TX_ROLLBACK_ONLY 2
```
4.3 Return Codes

Negative return values denote errors. An AP may regard non-negative return codes as denoting success, but these return codes may convey additional information.

```c
#define TX_NOT_SUPPORTED 1 /* normal execution */
#define TX_OK 0 /* normal execution */
#define TX_OUTSIDE -1 /* application is in an RM local transaction */
#define TX_ROLLBACK -2 /* transaction was rolled back */
#define TX_MIXED -3 /* transaction was partially committed and partially rolled back */
#define TX_HAZARD -4 /* transaction may have been partially committed and partially rolled back */
#define TX_PROTOCOL_ERROR -5 /* routine invoked in an improper context */
#define TX_ERROR -6 /* transient error */
#define TX_FAIL -7 /* fatal error */
#define TX_EINVAL -8 /* invalid arguments were given */
#define TX_COMMITTED -9 /* the transaction was heuristically committed */

#define TX_NO_BEGIN -100 /* transaction committed plus new transaction could not be started */
#define TX_ROLLBACK_NO_BEGIN (TX_ROLLBACK+TX_NO_BEGIN) /* transaction rollback plus new transaction could not be started */
#define TX_MIXED_NO_BEGIN (TX_MIXED+TX_NO_BEGIN) /* mixed plus transaction could not be started */
#define TX_HAZARD_NO_BEGIN (TX_HAZARD+TX_NO_BEGIN) /* hazard plus transaction could not be started */
#define TX_COMMITTED_NO_BEGIN (TX_COMMITTED+TX_NO_BEGIN) /* heuristically committed plus transaction could not be started */
```
This chapter describes the C interface to the TX service set. Reference manual pages appear, in alphabetical order, for each service in the TX interface. A TM provides these routines for APs to call.

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

In some descriptions there is a section entitled **Optional Set-up**; this lists those functions that might have been called and that govern the operation of the function being described.

In some cases the [TXFAIL] error description includes the following sentence:

The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application.

This means that any of the following can no longer perform work on behalf of the application:

- transaction manager
- any resource manager
- any combination of resource managers
- any combination of resource managers and the transaction manager.
NAME

 tx_begin — begin a global transaction

SYNOPSIS

#include <tx.h>
int tx_begin(void)

DESCRIPTION

The function tx_begin() is used to place the calling thread of control in transaction mode. The
calling thread must first ensure that its linked resource managers have been opened (by means
of tx_open()) before it can start transactions. The function tx_begin() fails (returning
[TX_PROTOCOL_ERROR]) if the caller is already in transaction mode or tx_open() has not been
called.

Once in transaction mode, the calling thread must call tx_commit() or tx_rollback() to complete
its current transaction. There are certain cases related to transaction chaining where tx_begin()
does not need to be called explicitly to start a transaction. See tx_commit() on page 23 and
tx_rollback() on page 27 for details.

Optional Set-up

• tx_set_transaction_timeout()

RETURN VALUE

Upon successful completion, tx_begin() returns [TX_OK], a non-negative return value.

ERRORS

Under the following conditions, tx_begin() fails and returns one of these negative values:

[TX_OUTSIDE]
The transaction manager is unable to start a global transaction because the calling thread of
control is currently participating in work outside any global transaction with one or more
resource managers. All such work must be completed before a global transaction can be
started. The caller's state with respect to the local transaction is unchanged.

[TX_PROTOCOL_ERROR]
The function was called in an improper context (for example, the caller is already in
transaction mode). The caller's state with respect to transaction mode is unchanged.

[TX_ERROR]
Either the transaction manager or one or more of the resource managers encountered a
transient error trying to start a new transaction. When this error is returned, the caller is not
in transaction mode. The exact nature of the error is determined in a product-specific
manner.

[TX_FAIL]
Either the transaction manager or one or more of the resource managers encountered a fatal
error. The nature of the error is such that the transaction manager and/or one or more of
the resource managers can no longer perform work on behalf of the application. When this
error is returned, the caller is not in transaction mode. The exact nature of the error is
determined in a product-specific manner.

APPLICATION USAGE

XA-compliant resource managers must be successfully opened to be included in the global
transaction. (See tx_open() on page 26 for details.)
SEE ALSO

    tx_commit(), tx_open(), tx_rollback(), tx_set_transaction_timeout().
NAME

    tx_close — close a set of resource managers

SYNOPSIS

    #include <tx.h>
    int tx_close(void)

DESCRIPTION

    The function tx_close() closes a set of resource managers in a portable manner. It invokes a
    transaction manager to read information specific to the resource manager in a manner specific to
    the transaction manager and pass this information to the resource managers linked to the caller.

    The function tx_close() closes all resource managers to which the caller is linked. This function
    is used in place of close calls specific to the resource manager and allows an application program
    to be free of calls, which may hinder portability. Since resource managers differ in their
    termination semantics, the specific information needed to close a particular resource manager
    must be published by each resource manager.

    The function tx_close() should be called when an application thread of control no longer wishes
    to participate in global transactions. The function tx_close() fails (returning
    [TX_PROTOCOL_ERROR]) if the caller is in transaction mode. That is, no resource managers
    are closed even though some may not be participating in the current transaction.

    When tx_close() returns success ([TX_OK]), all resource managers linked to the calling thread
    are closed.

RETURN VALUE

    Upon successful completion, tx_close() returns [TX_OK], a non-negative return value.

ERRORS

    Under the following conditions, tx_close() fails and returns one of these negative values:

    [TX_PROTOCOL_ERROR]
    The function was called in an improper context (for example, the caller is in transaction
    mode). No resource managers are closed.

    [TX_ERROR]
    Either the transaction manager or one or more of the resource managers encountered a
    transient error. The exact nature of the error is determined in a product-specific manner.
    All resource managers that could be closed are closed.

    [TX_FAIL]
    Either the transaction manager or one or more of the resource managers encountered a fatal
    error. The nature of the error is such that the transaction manager and/or one or more of
    the resource managers can no longer perform work on behalf of the application. The exact
    nature of the error is determined in a product-specific manner.

SEE ALSO

    tx_open().
NAME
   tx_commit — commit a global transaction

SYNOPSIS
   #include <tx.h>
   int tx_commit(void)

DESCRIPTION
   The function tx_commit() is used to commit the work of the transaction active in the caller's
   thread of control.

   If the transaction_control characteristic (see tx_set_transaction_control() on page 31) is
   TX_UNCHAINED, when tx_commit() returns, the caller is no longer in transaction mode.
   However, if the transaction_control characteristic is TX_CHAINED, when tx_commit() returns,
   the caller remains in transaction mode on behalf of a new transaction (see the RETURN VALUE
   and ERRORS sections below).

Optional Set-up
   • tx_set_commit_return()
   • tx_set_transaction_control()
   • tx_set_transaction_timeout()

RETURN VALUE
   Upon successful completion, tx_commit() returns [TX_OK], a non-negative return value.

ERRORS
   Under the following conditions, tx_commit() fails and returns one of these negative values:

   [TX_NO_BEGIN]
   The transaction committed successfully; however, a new transaction could not be started
   and the caller is no longer in transaction mode. This return value occurs only when the
   transaction_control characteristic is TX_CHAINED.

   [TX_ROLLBACK]
   The transaction could not commit and has been rolled back. In addition, if the
   transaction_control characteristic is TX_CHAINED, a new transaction is started.

   [TX_ROLLBACK_NO_BEGIN]
   The transaction could not commit and has been rolled back. In addition, a new transaction
   could not be started and the caller is no longer in transaction mode. This return value can
   occur only when the transaction_control characteristic is TX_CHAINED.

   [TX_MIXED]
   The transaction was partially committed and partially rolled back. In addition, if the
   transaction_control characteristic is TX_CHAINED, a new transaction is started.

   [TX_MIXED_NO_BEGIN]
   The transaction was partially committed and partially rolled back. In addition, a new
   transaction could not be started and the caller is no longer in transaction mode. This return
   value can occur only when the transaction_control characteristic is TX_CHAINED.

   [TX_HAZARD]
   Due to a failure, the transaction may have been partially committed and partially rolled
   back. In addition, if the transaction_control characteristic is TX_CHAINED, a new
   transaction is started.
Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX_CHAINED.

The function was called in an improper context (for example, the caller is not in transaction mode). The caller’s state with respect to the transaction is not changed.

Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner. The caller’s state with respect to the transaction is unknown.

SEE ALSO

tx_begin(), tx_set_commit_return(), tx_set_transaction_control(), tx_set_transaction_timeout().
NAME

`tx_info` — return global transaction information

SYNOPSIS

```c
#include <tx.h>
int tx_info(TXINFO *info)
```

DESCRIPTION

The function `tx_info()` returns global transaction information in the structure pointed to by `info`. In addition, this function returns a value indicating whether the caller is currently in transaction mode or not.

If `info` is non-null, `tx_info()` populates a `TXINFO` structure pointed to by `info` with global transaction information. The `TXINFO` structure contains the following elements:

```c
XID xid;
COMMIT_RETURN when_return;
TRANSACTION_CONTROL transaction_control;
TRANSACTION_TIMEOUT transaction_timeout;
TRANSACTION_STATE transaction_state;
```

If `tx_info()` is called in transaction mode, `xid` is populated with a current transaction branch identifier and `transaction_state` contains the state of the current transaction. If the caller is not in transaction mode, `xid` is populated with the null `XID` (see `<tx.h>` for details). In addition, regardless of whether the caller is in transaction mode, `when_return`, `transaction_control`, and `transaction_timeout` contain the current settings of the `commit_return` and `transaction_control` characteristics, and the transaction timeout value in seconds.

The transaction timeout value returned reflects the setting that is used when the next transaction is started. Thus, it may not reflect the timeout value for the caller’s current global transaction since calls made to `tx_set_transaction_timeout()` after the current transaction was begun may have changed its value.

If `info` is null, no `TXINFO` structure is returned.

RETURN VALUE

If the caller is in transaction mode, 1 is returned. If the caller is not in transaction mode, 0 is returned.

ERRORS

Under the following conditions, `tx_info()` fails and returns one of these negative values:

- `[TX_PROTOCOL_ERROR]`
  The function was called in an improper context (for example, the caller has not yet called `tx_open()`).

- `[TX_FAIL]`
  The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner.

APPLICATION USAGE

Within the same global transaction, subsequent calls to `tx_info()` are guaranteed to provide an `XID` with the same `gtrid` component, but not necessarily the same `bqual` component.

SEE ALSO

`tx_open()`, `tx_set_commit_return()`, `tx_set_transaction_control()`, `tx_set_transaction_timeout()`.
NAME

tx_open — open a set of resource managers

SYNOPSIS

#include <tx.h>

int tx_open(void)

DESCRIPTION

The function tx_open() opens a set of resource managers in a portable manner. It invokes a
transaction manager to read information specific to the resource manager in a manner specific to
the transaction manager and pass this information to the resource managers linked to the caller.

The function tx_open() attempts to open all resource managers that have been linked with the
application. This function is used in place of open calls specific to the resource manager and
allows an application program to be free of calls, which may hinder portability. Since resource
managers differ in their initialisation semantics, the specific information needed to open a
particular resource manager must be published by each resource manager.

If tx_open() returns [TX_ERROR], no resource managers are open. If tx_open() returns [TX_OK],
some or all of the resource managers have been opened. Resource managers that are not open
return errors specific to the resource manager when accessed by the application. The function
tx_open() must successfully return before a thread of control participates in global transactions.

Once tx_open() returns success, subsequent calls to tx_open() (before an intervening call to
tx_close()) are allowed. However, such subsequent calls return success, and the TM does not
attempt to reopen any RMs.

RETURN VALUE

Upon successful completion, tx_open() returns [TX_OK], a non-negative return value.

ERRORS

Under the following conditions, tx_open() fails and returns one of these negative values:

[TX_ERROR]

Either the transaction manager or one or more of the resource managers encountered a
transient error. No resource managers are open. The exact nature of the error is determined
in a product-specific manner.

[TX_FAIL]

Either the transaction manager or one or more of the resource managers encountered a fatal
error. The nature of the error is such that the transaction manager and/or one or more of
the resource managers can no longer perform work on behalf of the application. The exact
nature of the error is determined in a product-specific manner.

SEE ALSO

tx_close().
NAME

tx_rollback — roll back a global transaction

SYNOPSIS

#include <tx.h>
int tx_rollback(void)

DESCRIPTION

The function tx_rollback() is used to roll back the work of the transaction active in the caller's thread of control.

If the transaction_control characteristic (see tx_set_transaction_control() on page 31) is TX_UNCHAINED, when tx_rollback() returns, the caller is no longer in transaction mode. However, if the transaction_control characteristic is TX_CHAINED, when tx_rollback() returns, the caller remains in transaction mode on behalf of a new transaction (see the RETURN VALUE and ERRORS sections below).

Optional Set-up

• tx_set_transaction_control()
• tx_set_transaction_timeout()

RETURN VALUE

Upon successful completion, tx_rollback() returns [TX_OK], a non-negative return value.

ERRORS

Under the following conditions, tx_rollback() fails and returns one of these negative values:

[TX_NO_BEGIN]
The transaction rolled back; however, a new transaction could not be started and the caller is no longer in transaction mode. This return value occurs only when the transaction_control characteristic is TX_CHAINED.

[TX_MIXED]
The transaction was partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started.

[TX_MIXED_NO_BEGIN]
The transaction was partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX_CHAINED.

[TX_HAZARD]
Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started.

[TX_HAZARD_NO_BEGIN]
Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX_CHAINED.

[TX_COMMITTED]
The transaction was heuristically committed. In addition, if the transaction_control characteristic is TX_CHAINED, a new transaction is started.
[TX_COMMITTED_NO_BEGIN]
The transaction was heuristically committed. In addition, a new transaction could not be
started and the caller is no longer in transaction mode. This return value can occur only
when the transaction_control characteristic is TX_CHAINED.

[ TX_PROTOCOL_ERROR]
The function was called in an improper context (for example, the caller is not in transaction
mode).

[TX_FAIL]
Either the transaction manager or one or more of the resource managers encountered a fatal
error. The nature of the error is such that the transaction manager and/or one or more of
the resource managers can no longer perform work on behalf of the application. The exact
nature of the error is determined in a product-specific manner. The caller’s state with
respect to the transaction is unknown.

SEE ALSO
   tx_begin(), tx_set_transaction_control(), tx_set_transaction_timeout().
NAME

tx_set_commit_return — set commit_return characteristic

SYNOPSIS

#include <tx.h>

int tx_set_commit_return(COMMIT_RETURN when_return)

DESCRIPTION

The function tx_set_commit_return() sets the commit_return characteristic to the value specified in when_return. This characteristic affects the way tx_commit() behaves with respect to returning control to its caller. tx_set_commit_return() may be called regardless of whether its caller is in transaction mode. This setting remains in effect until changed by a subsequent call to tx_set_commit_return().

The initial setting for this characteristic is implementation dependent.

The valid settings for when_return are as follows:

TX_COMMIT_DECISION_LOGGED

This flag indicates that tx_commit() should return after the commit decision has been logged by the first phase of the two-phase commit protocol but before the second phase has completed. This setting allows for faster response to the caller of tx_commit(). However, there is a risk that a transaction has a heuristic outcome, in which case the caller does not find out about this situation by means of return codes from tx_commit(). Under normal conditions, participants that promise to commit during the first phase do so during the second phase. In certain unusual circumstances however (for example, long-lasting network or node failures) phase 2 completion may not be possible and heuristic results may occur. A transaction manager may optionally choose not to support this feature and may return [TX_NOT_SUPPORTED] to indicate that this value is not supported.

TX_COMMIT_COMPLETED

This flag indicates that tx_commit() should return after the two-phase commit protocol has finished completely. This setting allows the caller of tx_commit() to see return codes that indicate that a transaction had or may have had heuristic results. A transaction manager may optionally choose not to support this feature and may return [TX_NOT_SUPPORTED] to indicate that this value is not supported.

RETURN VALUE

Upon successful completion, tx_set_commit_return() returns [TX_OK], a non-negative return value. If the transaction manager does not support the setting of when_return to TX_COMMIT_COMPLETED or TX_COMMIT_DECISION_LOGGED, it returns [TX_NOT_SUPPORTED], a non-negative return value, and the commit_return characteristic remains set to its existing value. The transaction manager must support the setting of when_return to at least one of TX_COMMIT_COMPLETED or TX_COMMIT_DECISION_LOGGED.

ERRORS

Under the following conditions, tx_set_commit_return() does not change the setting of the commit_return characteristic and returns one of these negative values:

[TX EINVAL]

The argument when_return is not one of TX_COMMIT_DECISION_LOGGED or TX_COMMIT_COMPLETED.

[TX_PROTOCOL_ERROR]

The function was called in an improper context (for example, the caller has not yet called tx_open()).
[TX_FAIL]

The transaction manager encountered a fatal error. The nature of the error is such that the
transaction manager can no longer perform work on behalf of the application. The exact
nature of the error is determined in a product-specific manner.

SEE ALSO

\texttt{tx\_commit()}, \texttt{tx\_open()}, \texttt{tx\_info()}. 
NAME

`tx_set_transaction_control` — set `transaction_control` characteristic

SYNOPSIS

```c
#include <tx.h>
int tx_set_transaction_control(TRANSACTION_CONTROL control)
```

DESCRIPTION

The function `tx_set_transaction_control` sets the `transaction_control` characteristic to the value specified in `control`. This characteristic determines whether `tx_commit()` and `tx_rollback()` start a new transaction before returning to their caller. The function `tx_set_transaction_control` may be called regardless of whether the application program is in transaction mode. This setting remains in effect until changed by a subsequent call to `tx_set_transaction_control()`.

The initial setting for this characteristic is TX_UNCHAINED.

The valid settings for `control` are as follows:

- **TX_UNCHAINED**
  - This flag indicates that `tx_commit()` and `tx_rollback()` should not start a new transaction before returning to their caller. The caller must issue `tx_begin()` to start a new transaction.

- **TX_CHAINED**
  - This flag indicates that `tx_commit()` and `tx_rollback()` should start a new transaction before returning to their caller.

RETURN VALUE

Upon successful completion, `tx_set_transaction_control()` returns [TX_OK], a non-negative return value.

ERRORS

Under the following conditions, `tx_set_transaction_control()` does not change the setting of the `transaction_control` characteristic and returns one of these negative values:

- **[TX EINVAL]**
  - The argument `control` is not one of TX_UNCHAINED or TX_CHAINED.

- **[TX_PROTOCOL_ERROR]**
  - The function was called in an improper context (for example, the caller has not yet called `tx_open()`).

- **[TX_FAIL]**
  - The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

`tx_begin()`, `tx_commit()`, `tx_open()`, `tx_rollback()`, `tx_info()`.
NAME

tx_set_transaction_timeout — set transaction_timeout characteristic

SYNOPSIS

#include <tx.h>

int tx_set_transaction_timeout(TRANSACTION_TIMEOUT timeout)

DESCRIPTION

The function `tx_set_transaction_timeout()` sets the `transaction_timeout` characteristic to the value specified in `timeout`. This value specifies the time period in which the transaction must complete before becoming susceptible to transaction timeout; that is, the interval between the AP calling `tx_begin()` and `tx_commit()` or `tx_rollback()`. The function `tx_set_transaction_timeout()` may be called regardless of whether its caller is in transaction mode or not. If `tx_set_transaction_timeout()` is called in transaction mode, the new timeout value does not take effect until the next transaction.

The initial `transaction_timeout` value is 0 (no timeout).

The argument `timeout` specifies the number of seconds allowed before the transaction becomes susceptible to transaction timeout. It may be set to any value up to the maximum value for a type `long` as defined by the system. A `timeout` value of zero disables the timeout feature.

RETURN VALUE

Upon successful completion, `tx_set_transaction_timeout()` returns `[TX_OK]`, a non-negative return value.

ERRORS

Under the following conditions, `tx_set_transaction_timeout()` does not change the setting of the `transaction_timeout` characteristic and returns one of these negative values:

- `[TX EINVAL]`
  The timeout value specified is invalid.

- `[TX_PROTOCOL_ERROR]`
  The function was called in an improper context. For example, the caller has not yet called `tx_open()`.

- `[TX_FAIL]`
  The transaction manager encountered an error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

`tx_begin()`, `tx_commit()`, `tx_open()`, `tx_rollback()`, `tx_info()`. 
This chapter describes the X/Open COBOL interface to the TX service set. It is basically a one-to-one mapping of the calls, parameters and return codes, with minor differences due to the absence of function return in COBOL. This chapter assumes that a COBOL library for TX is written in C and that it uses the C interface to TX.

Reference manual pages appear, in alphabetical order, for each service in the TX interface. A TM provides these routines for APs to call.

In some descriptions there is a section entitled Optional Set-up; this lists those functions that might have been called and that govern the operation of the function being described.

In some cases the [TX-FAIL] error description includes the following sentence:

The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application.

This means that any of the following can no longer perform work on behalf of the application:

- transaction manager
- any resource manager
- any combination of resource managers
- any combination of resource managers and the transaction manager.
NAME

TXINTRO — COBOL data structures

DESCRIPTION

Overview

There is a syntactic description in COBOL for each call of the TX interface.

Each call is described by the following items:

• reference to the COBOL records in the Working-Storage Section needed by that call, by a COPY statement
• synopsis of the call in the Procedure Division
• description of the call
• list of the return codes.

Data Structures Used by the COBOL TX Interface

Two COBOL records: **TX-RETURN-STATUS** and **TX-INFO-AREA** are commonly used by the TX calls. They are expected to be defined in the Working-Storage Section by specification of COPY statements.

**TX-RETURN-STATUS**

Every function described in this chapter takes an instance of this record as a parameter. It is used to return a value to the caller. This record is expected to be used in the context:

```
01 TX-RETURN-STATUS COPY TXSTATUS.
```

**TXSTATUS** is a COBOL text library defining a signed integer that may be assigned one of the following values:

```
05 TX-STATUS PIC S9(9) COMP-5.
   88 TX-NOT-SUPPORTED VALUE 1.
   * Normal execution
   88 TX-OK VALUE 0.
   * Normal execution
   88 TX-OUTSIDE VALUE -1.
   * Application is in an RM local transaction
   88 TX-ROLLBACK VALUE -2.
   * Transaction was rolled back
   88 TX-MIXED VALUE -3.
   * Transaction was partially committed and partially rolled back
   88 TX-HAZARD VALUE -4.
   * Transaction may have been partially committed and partially rolled back
   88 TX-PROTOCOL-ERROR VALUE -5.
   * Routine invoked in an improper context
   88 TX-ERROR VALUE -6.
   * Transient error
   88 TX-FAIL VALUE -7.
   * Fatal error
   88 TX-EINVAL VALUE -8.
   * Invalid arguments were given
```
88 TX-COMMITTED VALUE -9.
* The transaction was heuristically committed
88 TX-NO-BEGIN VALUE -100.
* Transaction committed plus new transaction could not be started
88 TX-ROLLBACK-NO-BEGIN VALUE -102.
* Transaction rollback plus new transaction could not be started
88 TX-MIXED-NO-BEGIN VALUE -103.
* Mixed plus new transaction could not be started
88 TX-HAZARD-NO-BEGIN VALUE -104.
* Hazard plus new transaction could not be started
88 TX-COMMITTED-NO-BEGIN VALUE -109.
* Heuristically committed plus transaction could not be started

TX-INFO-AREA
This record defines a data structure where the result of the TXINFORM call is stored.

It is expected to be used in the context:

01 TX-INFO-AREA.
COPY TXINFDEF.

TXINFDEF is a COBOL text library defining a record as follows:

* XID record
  05 XID-REC.
   10 FORMAT-ID PIC S9(9) COMP-5.
   * A value of -1 in FORMAT-ID means that the XID is null
   10 GTRID-LENGTH PIC S9(9) COMP-5.
   10 BRANCH-LENGTH PIC S9(9) COMP-5.
   10 XID-DATA PIC X(128).

* Transaction mode settings
  05 TRANSACTION-MODE PIC S9(9) COMP-5.
   88 TX-NOT-IN-TRAN VALUE 0.
   88 TX-IN-TRAN VALUE 1.

* Commit_return settings
  05 COMMIT-RETURN PIC S9(9) COMP-5.
   88 TX-COMMIT-COMPLETED VALUE 0.
   88 TX-COMMIT-DECISION-LOGGED VALUE 1.

* Transaction_control settings
  05 TRANSACTION-CONTROL PIC S9(9) COMP-5.
   88 TX-UNCHAINED VALUE 0.
   88 TX-CHAINED VALUE 1.

* Transaction_timeout value
  05 TRANSACTION-TIMEOUT PIC S9(9) COMP-5.
   88 NO-TIMEOUT VALUE 0.

* Transaction_state information
  05 TRANSACTION-STATE PIC S9(9) COMP-5.
   88 TX-ACTIVE VALUE 0.
   88 TX-TIMEOUT-ROLLBACK-ONLY VALUE 1.
   88 TX-ROLLBACK-ONLY VALUE 2.
NAME
TXBEGIN — begin a global transaction

SYNOPSIS
DATA DIVISION.
* Include TX definitions.
01 TX-RETURN-STATUS.
COPY TXSTATUS.
PROCEDURE DIVISION.
CALL "TXBEGIN" USING TX-RETURN-STATUS.

DESCRIPTION
TXBEGIN is used to place the calling thread of control in transaction mode. The calling thread
must first ensure that its linked resource managers have been opened (by mean of TXOPEN)
before it can start transactions. TXBEGIN fails (with a TX-RETURN-STATUS value of [TX-
PROTOCOL-ERROR]) if the caller is already in transaction mode or TXOPEN has not been
called.

Once in transaction mode, the calling thread must call TXCOMMIT or TXROLLBACK to
complete its current transaction. There are certain cases related to transaction chaining where
TXBEGIN does not need to be called explicitly to start a transaction. See TXCOMMIT on page
39 and TXROLLBACK on page 44 for details.

Optional Set-up
  • TXSETTIMEOUT

RETURN VALUE
Upon successful completion, TXBEGIN sets [TX-OK], a non-negative return value.

ERRORS
Under the following conditions, TXBEGIN fails and sets one of these negative values:

[TX-OUTSIDE]
The transaction manager is unable to start a global transaction because the calling thread of
control is currently participating in work outside any global transaction with one or more
resource managers. All such work must be completed before a global transaction can be
started. The caller's state with respect to the local transaction is unchanged.

[TX-PROTOCOL-ERROR]
The function was called in an improper context (for example, the caller is already in
transaction mode). The caller's state with respect to transaction mode is unchanged.

[TX-ERROR]
Either the transaction manager or one or more of the resource managers encountered a
transient error trying to start a new transaction. When this error is returned, the caller is not
in transaction mode. The exact nature of the error is determined in a product-specific
manner.

[TX-FAIL]
Either the transaction manager or one or more of the resource managers encountered a fatal
error. The nature of the error is such that the transaction manager and/or one or more of
the resource managers can no longer perform work on behalf of the application. When this
error is returned, the caller is not in transaction mode. The exact nature of the error is
determined in a product-specific manner.
APPLICATION USAGE
   XA-compliant resource managers must be successfully opened to be included in the global transaction. (See TXOPEN on page 43 for details.)

SEE ALSO
   TXCOMMIT, TXOPEN, TXROLLBACK, TXSETTIMEOUT.
NAME

TXCLOSE — close a set of resource managers

SYNOPSIS

DATA DIVISION.

* Include TX definitions.

01 TX-RETURN-STATUS.

COPY TXSTATUS.

PROCEDURE DIVISION.

CALL "TXCLOSE" USING TX-RETURN-STATUS.

DESCRIPTION

TXCLOSE closes a set of resource managers in a portable manner. It invokes a transaction manager to read information specific to the resource manager in a manner specific to the transaction manager and pass this information to the resource managers linked to the caller.

TXCLOSE closes all resource managers to which the caller is linked. This function is used in place of close calls specific to the resource manager and allows an application program to be free of calls, which may hinder portability. Since resource managers differ in their termination semantics, the specific information needed to close a particular resource manager must be published by each resource manager.

TXCLOSE should be called when an application thread of control no longer wishes to participate in global transactions. TXCLOSE fails (returning [TX-PROTOCOL-ERROR]) if the caller is in transaction mode. That is, no resource managers are closed even though some may not be participating in the current transaction.

When TXCLOSE sets success ([TX-OK]), all resource managers linked to the calling thread are closed.

RETURN VALUE

Upon successful completion, TXCLOSE sets [TX-OK], a non-negative value.

ERRORS

Under the following conditions, TXCLOSE fails and sets one of these negative values:

[TX-PROTOCOL-ERROR]

The function was called in an improper context (for example, the caller is in transaction mode). No resource managers are closed.

[TX-ERROR]

Either the transaction manager or one or more of the resource managers encountered a transient error. The exact nature of the error is determined in a product-specific manner. All resource managers that could be closed are closed.

[TX-FAIL]

Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TXOPEN.
NAME
TXCOMMIT — commit a global transaction

SYNOPSIS
DATA DIVISION.
* Include TX definitions.
  01 TX-RETURN-STATUS.
      COPY TXSTATUS.
PROCEDURE DIVISION.
CALL "TXCOMMIT" USING TX-RETURN-STATUS.

DESCRIPTION
TXCOMMIT is used to commit the work of the transaction active in the caller’s thread of control. If the transaction_control characteristic (see TXSETTRANCTL on page 49) is TX-UNCHAINED, when TXCOMMIT returns, the caller is no longer in transaction mode. However, if the transaction_control characteristic is TX-CHAINED, when TXCOMMIT returns, the caller remains in transaction mode on behalf of a new transaction (see the RETURN VALUE and ERRORS sections below).

Optional Set-up
- TXSETCOMMITRET
- TXSETTRANCTL
- TXSETTIMEOUT

RETURN VALUE
Upon successful completion, TXCOMMIT sets [TX-OK], a non-negative return value.

ERRORS
Under the following conditions, TXCOMMIT fails and sets one of these negative values:

[TX-NO-BEGIN]
The transaction committed successfully; however, a new transaction could not be started and the caller is no longer in transaction mode. This return value occurs only when the transaction_control characteristic is TX-CHAINED.

[TX-ROLLBACK]
The transaction could not commit and has been rolled back. In addition, if the transaction_control characteristic is TX-CHAINED, a new transaction is started.

[TX-ROLLBACK-NO-BEGIN]
The transaction could not commit and has been rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX-CHAINED.

[TX-MIXED]
The transaction was partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX-CHAINED, a new transaction is started.

[TX-MIXED-NO-BEGIN]
The transaction was partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX-CHAINED.
[TX-HAZARD]
Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX-CHAINED, a new transaction is started.

[TX-HAZARD-NO-BEGIN]
Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX-CHAINED.

[TX-PROTOCOL-ERROR]
The function was called in an improper context (for example, the caller is not in transaction mode). The caller's state with respect to transaction mode is not changed.

[TX-FAIL]
Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner. The caller's state with respect to the transaction is unknown.

SEE ALSO
TXBEGIN, TXSETCOMMITRET, TXSETTRANCTL, TXSETTIMEOUT.
NAME
TXINFORM — return global transaction information

SYNOPSIS
DATA DIVISION.
* Include TX definitions.
  01 TX-RETURN-STATUS.
   COPY TXSTATUS.
*   01 TX-INFO-AREA.
   COPY TXINFDEF.
PROCEDURE DIVISION.
   CALL "TXINFORM" USING TX-INFO-AREA TX-RETURN-STATUS.

DESCRIPTION
TXINFORM sets global transaction information in TX-INFO-AREA. In addition, this function
sets a value indicating whether the caller is currently in transaction mode or not.
TXINFORM populates the TX-INFO-AREA record with global transaction information. The
contents of the TX-INFO-AREA record are described under TXINTRO on page 34.
If TXINFORM is called in transaction mode, TRANSACTION-MODE is set to TX-IN-TRAN,
XID-REC is populated with a current transaction branch identifier and TRANSACTION-STATE
contains the state of the current transaction. If the caller is not in transaction mode,
TRANSACTION-MODE is set to TX-NOT-IN-TRAN and XID-REC is populated with the null
XID (see TXINTRO on page 34 for details). In addition, regardless of whether the caller is in
transaction mode, COMMIT-RETURN, TRANSACTION-CONTROL, and TRANSACTION-
TIMEOUT contain the current settings of the commit_return and transaction_control
characteristics, and the transaction timeout value in seconds.

The transaction timeout value returned reflects the setting to be used when the next transaction
is started. Thus, it may not reflect the timeout value for the caller’s current global transaction
since calls made to TXSETTIMEOUT after the current transaction was begun may have changed
its value.

RETURN VALUE
Upon successful completion, TXINFORM sets [TX-OK], a non-negative return value.

ERRORS
Under the following conditions, TXINFORM fails and sets one of these negative values:
[TX-PROTOCOL-ERROR]
   The function was called in an improper context (for example, the caller has not yet called
   TXOPEN).
[TX-FAIL]
   The transaction manager encountered a fatal error. The nature of the error is such that the
   transaction manager can no longer perform work on behalf of the application. The exact
   nature of the error is determined in a product-specific manner.

APPLICATION USAGE
Within the same global transaction, subsequent calls to TXINFORM are guaranteed to provide
an XID with the same gtrid component, but not necessarily the same bqual component.
SEE ALSO
  TXOPEN, TXSETCOMMITRET, TXSETTRANCTL, TXSETTIMEOUT.
NAME

TXOPEN — open a set of resource managers

SYNOPSIS

DATA DIVISION.
* Include TX definitions.
01 TX-RETURN-STATUS.
   COPY TXSTATUS.
PROCEDURE DIVISION.
   CALL "TXOPEN" USING TX-RETURN-STATUS.

DESCRIPTION

TXOPEN opens a set of resource managers in a portable manner. It invokes a transaction manager to read information specific to the resource manager in a manner specific to the transaction manager and pass this information to the resource managers linked to the caller.

TXOPEN attempts to open all resource managers that have been linked with the application. This function is used in place of open calls specific to the resource manager and allows an application program to be free of calls, which may hinder portability. Since resource managers differ in their initialisation semantics, the specific information needed to open a particular resource manager must be published by each resource manager.

If TXOPEN sets [TX-ERROR], no resource managers are open. If TXOPEN sets [TX-OK], some or all of the resource managers have been opened. Resource managers that are not open return errors specific to the resource manager when accessed by the application. TXOPEN must successfully return before a thread of control participates in global transactions.

Once TXOPEN sets success, subsequent calls to TXOPEN (before an intervening call to TXCLOSE) are allowed. However, such subsequent calls return success, and the TM does not attempt to reopen any RMs.

RETURN VALUE

Upon successful completion, TXOPEN sets [TX-OK], a non-negative return value.

ERRORS

Under the following conditions, TXOPEN fails and sets one of these negative values:

[TX-ERROR]
Either the transaction manager or one or more of the resource managers encountered a transient error. No resource managers are open. The exact nature of the error is determined in a product-specific manner.

[TX-FAIL]
Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TXCLOSE.
NAME
TXROLLBACK — roll back a global transaction

SYNOPSIS
DATA DIVISION.
* Include TX definitions.
  01 TX-RETURN-STATUS.
     COPY TXSTATUS.
PROCEDURE DIVISION.
   CALL "TXROLLBACK" USING TX-RETURN-STATUS.

DESCRIPTION
TXROLLBACK is used to roll back the work of the transaction active in the caller’s thread of control.

If the transaction_control characteristic (see TXSETTRANCTL on page 49) is TX-UNCHAINED, when TXROLLBACK returns, the caller is no longer in transaction mode. However, if the transaction_control characteristic is TX-CHAINED, when TXROLLBACK returns, the caller remains in transaction mode on behalf of a new transaction (see the RETURN VALUE and ERRORS sections below).

Optional Set-up
• TXSETTRANCTL
• TXSETTIMEOUT

RETURN VALUE
Upon successful completion, TXROLLBACK sets [TX-OK], a non-negative return value.

ERRORS
Under the following conditions, TXROLLBACK fails and sets one of these negative values:

[TX-NO-BEGIN]
The transaction rolled back; however, a new transaction could not be started and the caller is no longer in transaction mode. This return value occurs only when the transaction_control characteristic is TX-CHAINED.

[TX-MIXED]
The transaction was partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX-CHAINED, a new transaction is started.

[TX-MIXED-NO-BEGIN]
The transaction was partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX-CHAINED.

[TX-HAZARD]
Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, if the transaction_control characteristic is TX-CHAINED, a new transaction is started.

[TX-HAZARD-NO-BEGIN]
Due to a failure, the transaction may have been partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX-CHAINED.
[TX-COMMITTED]
The transaction was heuristically committed. In addition, if the transaction_control characteristic is TX-CHAINED, a new transaction is started.

[TX-COMMITTED-NO-BEGIN]
The transaction was heuristically committed. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the transaction_control characteristic is TX-CHAINED.

[TX-PROTOCOL-ERROR]
The function was called in an improper context (for example, the caller is not in transaction mode).

[TX-FAIL]
Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner. The caller’s state with respect to the transaction is unknown.

SEE ALSO
TXBEGIN, TXSETTRANCTL, TXSETTIMEOUT.
NAME
TXSETCOMMITRET — set commit_return characteristic

SYNOPSIS
DATA DIVISION.
* Include TX definitions.
  01 TX-RETURN-STATUS.
    COPY TXSTATUS.
* 01 TX-INFO-AREA.
    COPY TXINFDEF.
PROCEDURE DIVISION.
CALL "TXSETCOMMITRET" USING TX-INFO-AREA TX-RETURN-STATUS.

DESCRIPTION
TXSETCOMMITRET sets the commit_return characteristic to the value specified in COMMIT-RETURN. This characteristic affects the way TXCOMMIT behaves with respect to returning control to its caller. TXSETCOMMITRET may be called regardless of whether its caller is in transaction mode. This setting remains in effect until changed by a subsequent call to TXSETCOMMITRET.

The initial setting for this characteristic is implementation dependent.

The valid settings for COMMIT-RETURN are as follows:

TX-COMMIT-DECISION-LOGGED
This flag indicates that TXCOMMIT should return after the commit decision has been logged by the first phase of the two-phase commit protocol but before the second phase has completed. This setting allows for faster response to the caller of TXCOMMIT. However, there is a risk that a transaction has a heuristic outcome, in which case the caller does not find out about this situation by means of return codes from TXCOMMIT. Under normal conditions, participants that promise to commit during the first phase do so during the second phase. In certain unusual circumstances however (for example, long-lasting network or node failures) phase 2 completion may not be possible and heuristic results may occur. A transaction manager may optionally choose not to support this feature and may return [TX-NOT-SUPPORTED] to indicate that this value is not supported.

TX-COMMIT-COMPLETED
This flag indicates that TXCOMMIT should return after the two-phase commit protocol has finished completely. This setting allows the caller of TXCOMMIT to see return codes that indicate that a transaction had or may have had heuristic results. A transaction manager may optionally choose not to support this feature and may return [TX-NOT-SUPPORTED] to indicate that this value is not supported.

RETURN VALUE
Upon successful completion, TXSETCOMMITRET sets [TX-OK], a non-negative return value. If the transaction manager does not support the setting of COMMIT-RETURN to TX-COMMIT-COMPLETED or TX-COMMIT-DECISION-LOGGED, it returns [TX-NOT-SUPPORTED], a non-negative return value, and the commit_return characteristic remains set to its existing value. The transaction manager must support the setting of COMMIT-RETURN to at least one of TX-COMMIT-COMPLETED or TX-COMMIT-DECISION-LOGGED.
ERRORS

Under the following conditions, TXSETCOMMITRET does not change the setting of the commit_return characteristic and sets one of these negative values:

[TX-EINVAL]

   COMMIT-RETURN is not one of TX-COMMIT-DECISION-LOGGED or TX-COMMIT-COMPLETED.

[TX-PROTOCOL-ERROR]

   The function was called in an improper context (for example, the caller has not yet called TXOPEN).

[TX-FAIL]

   The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

   TXCOMMIT, TXOPEN, TXINFORM.
NAME
TXSETTIMEOUT — set transaction_timeout characteristic

SYNOPSIS
DATA DIVISION.
* Include TX definitions.
  01 TX-RETURN-STATUS.
     COPY TXSTATUS.
*
  01 TX-INFO-AREA.
     COPY TXINFDEF.
PROCEDURE DIVISION.
   CALL "TXSETTIMEOUT" USING TX-INFO-AREA TX-RETURN-STATUS.

DESCRIPTION
TXSETTIMEOUT sets the transaction_timeout characteristic to the value specified in
TRANSACTION-TIMEOUT. This value specifies the time period in which the transaction
must complete before becoming susceptible to transaction timeout; that is, the interval between
the AP calling TXBEGIN and TXCOMMIT or TXROLLBACK. TXSETTIMEOUT may be called
regardless of whether its caller is in transaction mode or not. If TXSETTIMEOUT is called in
transaction mode, the new timeout value does not take effect until the next transaction.

The initial transaction_timeout value is 0 (no timeout).

TRANSACTION-TIMEOUT specifies the number of seconds allowed before the transaction
becomes susceptible to transaction timeout. It may be set to any value up to the maximum value
for an S9(9) COMP 5 as defined by the system. A TRANSACTION-TIMEOUT value of zero
disables the timeout feature.

RETURN VALUE
Upon successful completion, TXSETTIMEOUT sets [TX-OK], a non-negative return value.

ERRORS
Under the following conditions, TXSETTIMEOUT does not change the setting of the
transaction_timeout characteristic and sets one of these negative values:

[TX-EINVAL]
The timeout value specified is invalid.

[TX-PROTOCOL-ERROR]
The function was called in an improper context. For example, the caller has not yet called
TXOPEN.

[TX-FAIL]
The transaction manager encountered an error. The nature of the error is such that the
transaction manager can no longer perform work on behalf of the application. The exact
nature of the error is determined in a product-specific manner.

SEE ALSO
TXBEGIN, TXCOMMIT, TXOPEN, TXROLLBACK, TXINFORM.
NAME

TXSETTRANCTL — set transaction_control characteristic

SYNOPSIS

DATA DIVISION.

* Include TX definitions.

01 TX-RETURN-STATUS.
   COPY TXSTATUS.

* 01 TX-INFO-AREA.
   COPY TXINFDEF.

PROCEDURE DIVISION.

CALL "TXSETTRANCTL" USING TX-INFO-AREA TX-RETURN-STATUS.

DESCRIPTION

TXSETTRANCTL sets the transaction_control characteristic to the value specified in TRANSACTION-CONTROL. This characteristic determines whether TXCOMMIT and TXROLLBACK start a new transaction before returning to their caller. TXSETTRANCTL may be called regardless of whether the application program is in transaction mode. This setting remains in effect until changed by a subsequent call to TXSETTRANCTL.

The initial setting for this characteristic is TX-UNCHAINED.

The valid settings for TRANSACTION-CONTROL are as follows:

TX-UNCHAINED
   This flag indicates that TXCOMMIT and TXROLLBACK should not start a new transaction before returning to their caller. The caller must issue TXBEGIN to start a new transaction.

TX-CHAINED
   This flag indicates that TXCOMMIT and TXROLLBACK should start a new transaction before returning to their caller.

RETURN VALUE

Upon successful completion, TXSETTRANCTL sets [TX-OK], a non-negative return value.

ERRORS

Under the following conditions, TXSETTRANCTL does not change the setting of the transaction_control characteristic and sets one of these negative values:

[TX-EINVAL]
   TRANSACTION-CONTROL is not one of TX-UNCHAINED or TX-CHAINED.

[TX-PROTOCOL-ERROR]
   The function was called in an improper context (for example, the caller has not yet called TXOPEN).

[TX-FAIL]
   The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is determined in a product-specific manner.

SEE ALSO

TXBEGIN, TXCOMMIT, TXOPEN, TXROLLBACK, TXINFORM.
This chapter contains a state table that shows legal calling sequences for the C-language TX routines for the transaction initiator.

The TM ensures that each thread of control calls the TX routines in a legal sequence. When an illegal state transition is attempted (that is, a call from a state with a blank transition entry), the called function returns [TX_PROTOCOL_ERROR]. The legal states and transitions for the TX primitives for the transaction initiator are shown in the table below. Calls that return failure do not make state transitions, except where described by specific state table entries.

The states are defined below:

S0  No RMs have been opened or initialised. An application thread of control cannot start a global transaction until it has successfully opened its RMs via tx_open().

S1  The thread has opened its RMs but is not in a transaction. Its transaction_control characteristic is TX_UNCHAINED.

S2  The thread has opened its RMs but is not in a transaction. Its transaction_control characteristic is TX_CHAINED.

S3  The thread has opened its RMs and is in a transaction. Its transaction_control characteristic is TX_UNCHAINED.

S4  The thread has opened its RMs and is in a transaction. Its transaction_control characteristic is TX_CHAINED.

The valid states for each function are as shown in Table 7-1 on page 52.
<table>
<thead>
<tr>
<th>Function</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_begin()</td>
<td>S0 S3 S4 S4</td>
</tr>
<tr>
<td>tx_close()</td>
<td>S0 S0 S0 S4</td>
</tr>
<tr>
<td>tx_commit() \rightarrow TX_SET1</td>
<td>S0 S0 S1 S4</td>
</tr>
<tr>
<td>tx_commit() \rightarrow TX_SET2</td>
<td>S0 S0 S2 S4</td>
</tr>
<tr>
<td>tx_info()</td>
<td>S1 S1 S2 S3</td>
</tr>
<tr>
<td>tx_open()</td>
<td>S1 S1 S2 S3</td>
</tr>
<tr>
<td>tx_rollback() \rightarrow TX_SET1</td>
<td>S1 S1 S2 S4</td>
</tr>
<tr>
<td>tx_rollback() \rightarrow TX_SET2</td>
<td>S1 S2 S3 S4</td>
</tr>
<tr>
<td>tx_set_commit_return()</td>
<td>S1 S2 S3 S4</td>
</tr>
<tr>
<td>tx_set_transaction_control()</td>
<td>S1 S1 S2 S3</td>
</tr>
<tr>
<td>control = TX_CHAINED</td>
<td>S1 S1 S2 S3</td>
</tr>
<tr>
<td>tx_set_transaction_timeout()</td>
<td>S1 S1 S2 S3</td>
</tr>
</tbody>
</table>

Table 7-1 C-language State Table

Notes:

1. TX_SET1 denotes any of:
   - [TX_OK]
   - [TX_ROLLBACK]
   - [TX_MIXED]
   - [TX_HAZARD]
   - [TX_COMMITTED]

   ([TX_ROLLBACK] is not returned by tx_rollback() and [TX_COMMITTED] is not returned by tx_commit()).

2. TX_SET2 denotes any of:
   - [TX_NO_BEGIN]
   - [TX_ROLLBACK_NO_BEGIN]
   - [TX_MIXED_NO_BEGIN]
   - [TX_HAZARD_NO_BEGIN]
   - [TX_COMMITTED_NO_BEGIN]

   ([TX_ROLLBACK_NO_BEGIN] is not returned by tx_rollback() and [TX_COMMITTED_NO_BEGIN] is not returned by tx_commit()).

3. If [TX_FAIL] is returned on any call, the application thread of control is in an undefined state with respect to the above table.

4. When tx_info() returns one of:
   - [TX_ROLLBACK_ONLY]
   - [TX_TIMEOUT_ROLLBACK_ONLY]

   in the transaction state information, the transaction is marked rollback-only and is rolled back whether the application program calls tx_commit() or tx_rollback().

52 X/Open CAE Specification
Chapter 8

Implementation Requirements

This chapter summarises the implications on implementors of this specification. It also identifies features of this specification that implementors of TMs, or application writers, can regard as optional.

These requirements are designed to facilitate portability — specifically, the ability to move an application program 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:

• relinking object modules

• supplying text strings to the software components (or executing a vendor-supplied procedure that incorporates suitable text strings).

For additional implementation requirements, see the XA specification.

8.1 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 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 operating without a TM, except that it calls the TM to define global transactions and to open and close RMs via the TX interface specified herein.

8.2 Resource Manager Requirements

The X/Open DTP model affects only RMs operating in the DTP environment. The model puts minimal constraints on the native interface by which APs request work from the RM.

Transaction demarcation with a particular RM is coordinated by the associated TM, using the XA interface (see the XA specification, which specifies information that an RM product must publish).
8.3 Transaction Manager Requirements

- **Service interfaces**
  TMs must support interaction with RMs using the XA interface.
  TMs must implement certain calls in the TX interface by making XA calls to each relevant RM, as specified in Appendix B. When one or more RMs report errors on those XA calls, the TM must convey appropriate error reports to the AP, as specified in that appendix.

- **Public information**
  TMs must publish their behaviour with respect to setting of the `commit_return` characteristic (specifically the default value and which settings are supported).
  See also the XA specification for details of the publication requirements for TMs.
Appendix A

<tx.h> C Header

This appendix specifies a <tx.h> C header file in both ISO C and Common Usage C. Any <tx.h> file used must contain at least the components of the file shown below, except for the comments.

/*
 * Start of tx.h header
 *
 * Define a symbol to prevent multiple inclusions of this header file
 */

#ifndef TX_H
#define TX_H

#define TX_H_VERSION 0 /* current version of this header file */

/*
 * Transaction identifier
 */

#define XIDDATASIZE 128 /* size in bytes */
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.
 */

/*
 * Definitions for tx_*() routines
 */

/* commit return values */
typedef long COMMIT_RETURN;
#define TX_COMMIT_COMPLETED 0
#define TX_COMMIT_DECISION_LOGGED 1

/* transaction control values */
typedef long TRANSACTION_CONTROL;
#define TX_UNCHAINED 0
#define TX_CHAINED 1

/* type of transaction timeouts */
typedef long TRANSACTION_TIMEOUT;

/* transaction state values */
typedef long TRANSACTION_STATE;
#define TX_ACTIVE 0
#define TX_TIMEOUTROLLBACK_ONLY 1
#define TX_ROLLBACKONLY 2

/* structure populated by tx_info() */
struct tx_info_t {
    XID xid;
    COMMIT_RETURN when_return;
    TRANSACTION_CONTROL transaction_control;
    TRANSACTION_TIMEOUT transaction_timeout;
    TRANSACTION_STATE transaction_state;
};
typedef struct tx_info_t TXINFO;

/*
 * Declarations of routines by which Applications call TMs:
 */
#endif /* ifndef __STDC__ */
#endif /* ifndef __STDC__ */

#define TX_NOT_SUPPORTED 1 /* option not supported */
#define TX_OK 0 /* normal execution */
#define TX_OUTSIDE -1 /* application is in an RM local transaction */
#define TX_ROLLBACK -2 /* transaction was rolled back */
#define TX_MIXED -3 /* transaction was partially committed and partially rolled back */
#define TX_HAZARD -4 /* transaction may have been partially committed and partially rolled back */
#define TX_PROTOCOL_ERROR -5 /* routine invoked in an improper context */
#define TX_ERROR -6 /* transient error */
#define TX_FAIL -7 /* fatal error */
#define TX_EINVAL -8 /* invalid arguments were given */
#define TX_COMMITTED -9 /* transaction has heuristically committed */
#define TX_NO_BEGIN -100 /* transaction committed plus new transaction could not be started */
#define TX_ROLLBACK_NO_BEGIN (TX_ROLLBACK+TX_NO_BEGIN) /* transaction rollback plus new transaction could not be started */
#define TX_MIXED_NO_BEGIN (TX_MIXED+TX_NO_BEGIN) /* mixed plus new transaction could not be started */
#define TX_HAZARD_NO_BEGIN (TX_HAZARD+TX_NO_BEGIN) /* hazard plus new transaction could not be started */
#define TX_COMMITTED_NO_BEGIN (TX_COMMITTED+TX_NO_BEGIN) /* heuristically committed plus new transaction could not be started */

#endif /* ifndef TX_H */

/ * End of tx.h header */
This appendix suggests mappings between the TX interface and the XA interface for the C-language function calls and C-language return codes.

The XA interface is sufficiently rich to permit alternative mappings in some cases. These alternatives are not shown in this appendix. Therefore, the tables shown in this appendix are not state tables.

B.1 Overview

The TX interface in this document specifies the calls an AP makes to the TM. X/Open-compliant TMs use the XA interface published in the XA specification to instruct all relevant RMs linked with the AP to carry out the functions expressed by the semantics of the TX interface. Most TX calls map to XA calls, and XA return codes map to TX return codes. When an AP thread of control issues such a TX call, the TM issues the associated call or calls in the XA interface for each relevant RM as suggested below.
B.2 Function Call Mappings

TX calls that result in \texttt{xa_*} calls could result in calls to \texttt{xa_complete()}. The TM has the option of using the asynchronous calling mode of XA, if the RM supports it. However, the asynchronous XA operation and the call or calls to \texttt{xa_complete()} that detect the operation's completion must occur wholly within the single TX call.

Some TX calls (for example, \texttt{tx_open()}) could result in calls to \texttt{xa_recover()}, \texttt{xa_commit()}, \texttt{xa_rollback()}, and \texttt{xa_forget()}, if the TM uses the TX call as an opportunity to recover or advance transaction work performed previously.

\texttt{tx_open()}
\rightarrow \texttt{xa_open()}

This call opens all RMs linked with the AP. The TM calls \texttt{xa_open()} at all RMs that are linked with the thread only for the first call to \texttt{tx_open()} in that thread of control, or for the first call to \texttt{tx_open()} after a call to \texttt{tx_close()}.

An RM that has received an \texttt{xa_open()} call in a given thread of control and responded with [XA_OK] may participate in global transactions in that thread.

\texttt{tx_begin()}
\rightarrow \texttt{xa_start()}

This call associates the AP thread of control with a global transaction. The TM generates an \texttt{XID} and calls \texttt{xa_start()} at all open RMs that are linked with the thread. However, the TM does not call \texttt{xa_start()} for dynamically-registering RMs; they register with the TM by calling \texttt{ax_reg()} only if the AP calls them through the native interface to request actual work.

An RM that has received an \texttt{xa_start()} call, or made an \texttt{ax_reg()} call, in a given thread of control for a given global transaction is said to be \texttt{associated} with the global transaction.

\texttt{tx_commit()}
\rightarrow \texttt{xa_end()}, \texttt{xa_prepare()}, \texttt{xa_commit()}
\rightarrow \texttt{xa_rollback()} if any RM vetos the transaction

The AP calls \texttt{tx_commit()} to make the effects of the transaction permanent. The TM, in turn, executes the two-phase commit protocol. The TM first calls \texttt{xa_end()} for each involved RM, from the AP's thread of control, to dissociate the thread from the global transaction. Then the TM coordinates the transaction commitment protocol as follows:

- **Phase 1**: The TM calls \texttt{xa_prepare()} for each RM that was associated with the global transaction (as defined above under \texttt{tx_begin()}). The \texttt{XA} specification describes techniques, such as an asynchronous calling mode, by which the TM may be able to schedule concurrent activities efficiently at different RMs.

- **Phase 2**: If all such RMs return success from \texttt{xa_prepare()}, the TM stably records a decision to commit the transaction and then calls \texttt{xa_commit()} for each such RM. If any RM returns error during Phase 1, then the TM calls \texttt{xa_rollback()} for each such RM that has not already rolled back its work, and \texttt{tx_commit()} returns to the AP a rollback indication.

The \texttt{XA} specification gives two optimisations in this procedure:

1. During Phase 1, any RM may report that the AP did not ask it to update shared resources, which eliminates the Phase 2 call to that RM.

2. If the TM has dealt with only one subordinate RM in the global transaction, it can omit Phase 1.
→ *xa_start()* if the AP’s *transaction_control* characteristic is set to TX_CHAINED.

See the description of *xa_start()* under *tx_begin()* above for details.

→ *xa_forget()* if any RM reports heuristics

If an RM informs the TM of a heuristic decision or heuristic hazard in response to *tx_commit()* or *xa_rollback()* , the TM must call *xa_forget()* to authorise the RM to discard knowledge of the global transaction. The TM can make this call before it returns from *tx_commit()* or arrange for the call to be made later.

*tx_rollback()*

→ *xa_end()*

→ *xa_rollback()*

The AP calls *tx_rollback()* to roll back the global transaction. As with *tx_commit()* , the TM first calls *xa_end()* for each RM to dissociate the AP thread of control from the transaction. Then the TM calls *xa_rollback()* for every RM that has not already rolled back its work.

→ *xa_start()* if the AP’s *transaction_control* characteristic is set to TX_CHAINED.

See the description of *xa_start()* under *tx_begin()* above for details.

→ *xa_forget()* if any RM reports heuristics

If an RM informs the TM of a heuristic decision or heuristic hazard in response to *xa_rollback()* , the TM must call *xa_forget()* to authorise the RM to discard knowledge of the global transaction.

*tx_close()*

→ *xa_close()*

This call closes all RMs linked with the AP. The TM calls *xa_close()* at all opened RMs that are linked with the thread.

An RM that has received an *xa_close()* call in a given thread of control may no longer participate in global transactions in that thread.

*tx_info()*

→ No corresponding *xa_* calls.

*tx_set_commit_return()*

→ No corresponding *xa_* calls.

*tx_set_transaction_control()*

→ No corresponding *xa_* calls.

*tx_set_transaction_timeout()*

→ No corresponding *xa_* calls.
B.3 General Rules for Mapping of Return Codes

These rules are implicit in all of the mapping tables that follow. Section B.4 on page 64 describes mappings from XA error codes to TX error codes in the case where only one RM is involved. The case where several RMs are involved and return different results is covered in Section B.5 on page 67.

Reporting Success

The RM return code \([\text{XA_OK}]\) maps directly to the TM return code \([\text{TX_OK}]\), and is not included in the tables. However, the notion of success depends on the context of the call:

- If an AP requests \(\text{tx\_rollback}()\) and all RMs roll back their work, \(\text{tx\_rollback}()\) reports success despite any RM failures or heuristic decisions that may have occurred.

- Conversely, if an AP requests \(\text{tx\_commit}()\) and the TM determines that it must call \(\text{xa\_rollback}()\) at some RMs, then despite the fact that these rollbacks succeeded, the TM must still return \([\text{TX\_ROLLBACK}]\) to notify the AP that the requested commitment failed.

- The \(\text{tx\_commit}()\) and \(\text{tx\_rollback}()\) calls typically cause the TM to issue a series of XA calls. Success at any one of these calls does not constitute success of the TX call. This is described in more detail later in this appendix.

Origin of Error

The return code \([\text{TX\_ERROR}]\) reports that an RM failed temporarily. The exact nature of the error may be reported in an RM-specific manner.

The return code \([\text{TX\_FAIL}]\) reports that the TM or an RM failed and the TM should not be called further.

In a variety of synchronisation failures between the RM and TM, it is important and not always clear which component experienced the failure. A TM should return \([\text{TX\_FAIL}]\) only when it can no longer perform work on behalf of the AP, otherwise it should return \([\text{TX\_ERROR}]\).

Heuristic Outcomes

As described in Section 3.3.1 on page 11, heuristic outcomes are cases where RMs make autonomous commitment decisions (see also the referenced DTP guide). Heuristic outcomes that match the outcome the AP requested (namely, \(\text{XA\_HEURCOM}\) during a call to \(\text{tx\_commit}()\), and \(\text{XA\_HEURRB}\) during a call to \(\text{tx\_rollback}()\)) are not reported to the AP as heuristic outcomes.

Invalid Arguments

The return code \([\text{XAER\_INVAL}]\) in the XA interface lets RMs report to the TM that the TM supplied an invalid argument. This never reflects an application coding error, because there is no TX routine where the AP gives the TM an argument to pass directly to the RM. \([\text{XAER\_INVAL}]\) reflects a synchronisation failure between RM and TM (see Origin of Error).
Protocol Violations

The return code [XAER_PROTO] in the XA interface lets RMs report to the TM that an XA routine was called in an improper context. TMs are required to enforce proper sequencing of calls to TX routines. When an improperly-coded AP causes a protocol error, the TM returns [TX_PROTOCOL_ERROR], for example, on the second consecutive call to `tx_begin()`. Therefore, [XAER_PROTO] reflects a synchronisation failure between an RM and a TM (see Origin of Error on page 62).

Too Many Asynchronous Operations

An asynchronous calling mode in the XA interface lets the TM efficiently schedule concurrent activities at different RMs. Support for the calling mode depends on the RM. The return code [XAER_ASYNC] informs the TM that it has exceeded the RM’s limit for outstanding asynchronous requests (which may be 0). The TM adapts to this case in a product-specific manner. The return code [XAERASYNC] is not included in the following tables.
### B.4 Suggested Mapping of Return Codes: Single RM

This section suggests a mapping of the return codes for a single RM where the TM employs a one-phase commit optimisation.

#### Return Codes from `xa_open()`

The TM opens the RM when the AP calls `tx_open()`.

<table>
<thead>
<tr>
<th><code>xa_open()</code></th>
<th><code>tx_open()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[XAER_RMERR]</td>
<td>[TX_ERROR]</td>
</tr>
<tr>
<td>[XAER_INVAL]</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAER_PROTO]</td>
<td>[TX_FAIL]</td>
</tr>
</tbody>
</table>

#### Return Codes from `xa_close()`

The TM closes the RM when the AP calls `tx_close()`.

<table>
<thead>
<tr>
<th><code>xa_close()</code></th>
<th><code>tx_close()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[XAER_RMERR]</td>
<td>[TX_ERROR]</td>
</tr>
<tr>
<td>[XAER_INVAL]</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAER_PROTO]</td>
<td>[TX_FAIL]</td>
</tr>
</tbody>
</table>

#### Return Codes from `xa_start()`

The TM starts a new global transaction at the RM when the AP calls `tx_begin()`, or when the AP calls `tx_commit()` or `tx_rollback()` and the `transaction_control` characteristic is set to `TX_CHAINED`.

<table>
<thead>
<tr>
<th><code>xa_start()</code></th>
<th><code>tx_begin()</code></th>
<th><code>tx_commit()</code></th>
<th><code>tx_rollback()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[XA_RETRY]</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>[XA_RB*]</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>[XAER_NOTA]</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>[XAER_RMERR]</td>
<td>[TX_ERROR]</td>
<td>[TX_*_NO_BEGIN] 4</td>
<td>[TX_*_NO_BEGIN] 5</td>
</tr>
<tr>
<td>[XAER_DUPID]</td>
<td>[TX_ERROR]</td>
<td>[TX_*_NO_BEGIN] 3,4</td>
<td>[TX_*_NO_BEGIN] 3,5</td>
</tr>
<tr>
<td>[XAER_INVAL]</td>
<td>[TX_FAIL]</td>
<td>[TX_*_NO_BEGIN] 4</td>
<td>[TX_*_NO_BEGIN] 5</td>
</tr>
<tr>
<td>[XAER_PROTO]</td>
<td>[TX_ERROR]</td>
<td>[TX_*_NO_BEGIN] 3,4</td>
<td>[TX_*_NO_BEGIN] 3,5</td>
</tr>
<tr>
<td>[XAER_RMFAIL]</td>
<td>[TX_FAIL]</td>
<td>[TX_*_NO_BEGIN] 4</td>
<td>[TX_*_NO_BEGIN] 5</td>
</tr>
<tr>
<td>[XAER_OUTSIDE]</td>
<td>[TX_OUTSIDE]</td>
<td>[TX_*_NO_BEGIN] 6</td>
<td>[TX_*_NO_BEGIN] 6</td>
</tr>
</tbody>
</table>

#### Notes:

1. The `[XA_RETRY]` code tells the TM to reissue the call. The result the TM returns to the AP depends on the ultimate outcome of the reissued call. If the RM keeps returning `[XA_RETRY]`, the TM may return `[TX_ERROR]`.

2. These return codes indicate that the global transaction has been marked rollback-only. An RM would never return them when the TM calls `xa_start()` to start a new global transaction. An RM only returns these codes when a thread of control uses the `xa_start(TMRESUME)` or `xa_start(TMJOIN)` forms to resume or join an existing global transaction that the RM has marked rollback-only. This situation is not applicable here.
3. Receipt of [XAER DUPID] or [XAER PROTO] is a strong indication of a failure between the TM and the RM. The TM may try to resynchronise with the RM by issuing `xa_end()` and `xa_rollback()`. If this is unsuccessful, [TX_FAIL] may be returned to the AP. See Origin of Error on page 62.

4. May be any one of [TX NO BEGIN], [TX ROLLBACK NO BEGIN].

5. May be any one of [TX NO BEGIN], [TX COMMITTED NO BEGIN].

6. This can only occur if the AP is participating in a local transaction. Since an AP cannot at the same time be within a global and a local transaction, this situation is not applicable here.

Return Codes from `xa_end()`

This section describes return codes from `xa_end()` when the TM calls it during `tx_commit()` or `tx_rollback()`.

<table>
<thead>
<tr>
<th><code>xa_end()</code></th>
<th><code>tx_commit()</code> or <code>tx_rollback()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[XA NOMIGRATE]</td>
<td>1</td>
</tr>
<tr>
<td>[XA RB*]</td>
<td>2</td>
</tr>
<tr>
<td>[XAER NOTA]</td>
<td>2</td>
</tr>
<tr>
<td>[XAER RMERR]</td>
<td>2</td>
</tr>
<tr>
<td>[XAER RMFAIL]</td>
<td>[TX FAIL]</td>
</tr>
<tr>
<td>[XAER INVAL]</td>
<td>[TX FAIL]</td>
</tr>
<tr>
<td>[XAER PROTO]</td>
<td>[TX FAIL]</td>
</tr>
</tbody>
</table>

Notes:

1. The RM returns [XA NOMIGRATE] only when the TM uses the form `xa_end(TMSUSPEND)`, which is not applicable here.

2. These return codes indicate to the TM that the RM in question has requested the TM to roll back all work on behalf of a global transaction. If the AP called `tx_commit()`, it receives [TX ROLLBACK]; if the AP called `tx_rollback()`, it receives [TX_OK].

3. Receipt of [XAER PROTO] is a strong indication of a failure between the TM and the RM. The TM may try to resynchronise with the RM by issuing `xa_rollback()`. If this is unsuccessful, [TX_FAIL] may be returned to the AP. See Origin of Error on page 62.
Return Codes from `xa_commit()` and `xa_rollback()`

This section describes return codes from `xa_commit()` and `xa_rollback()` when called by the TM during `tx_commit()` or `tx_rollback()`.

<table>
<thead>
<tr>
<th><code>xa_commit()</code> or <code>xa_rollback()</code></th>
<th><code>tx_commit()</code> or <code>tx_rollback()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[XA_HEURCOM] (\rightarrow) 1</td>
<td></td>
</tr>
<tr>
<td>[XA_HEURRB] (\rightarrow) 1</td>
<td></td>
</tr>
<tr>
<td>[XA_HEURHAZ] (\rightarrow) [TX_HAZARD]</td>
<td></td>
</tr>
<tr>
<td>[XA_HEURMIX] (\rightarrow) [TX_Mixed]</td>
<td></td>
</tr>
<tr>
<td>[XA_RB*] (\rightarrow) 2</td>
<td></td>
</tr>
<tr>
<td>[XAER_RMERR] (\rightarrow) 2</td>
<td></td>
</tr>
<tr>
<td>[XAER_NOTA] (\rightarrow) 2</td>
<td></td>
</tr>
<tr>
<td>[XAER_RMFALL] (\rightarrow) [TX_FAIL]</td>
<td></td>
</tr>
<tr>
<td>[XAERINVAL] (\rightarrow) [TX_FAIL]</td>
<td></td>
</tr>
<tr>
<td>[XAER_PROTO] (\rightarrow) [TX_FAIL]</td>
<td></td>
</tr>
<tr>
<td>[XA_RETRY] (\rightarrow) 3</td>
<td></td>
</tr>
</tbody>
</table>

Notes:

1. If a heuristic outcome matches the outcome the AP requested, the TM reports success ([TX_OK]) to the AP. If the outcome is not what the AP requested, the TM reports that disposition to the AP: `tx_commit()` returns [TX_ROLLBACK]; `tx_rollback()` returns [TX_COMMITTED].

2. These return codes indicate to the TM that the RM in question rolled back work it did on behalf of a global transaction. If the AP called `tx_commit()`, it receives [TX_ROLLBACK]; if the AP called `tx_rollback()`, it receives [TX_OK].

3. The XA_RETRY code tells the TM to reissue the call. The result the TM returns to the AP depends on the ultimate outcome of the reissued call. If the RM keeps returning XA_RETRY, the TM may return [TX_ERROR].
B.5  Suggested Mapping of Return Codes: Multiple RMs

The TM considers return status from all associated RMs to generate a return code for the AP. In general, the TM’s return code reflects the *most severe error* an RM reported to it. The suggested hierarchy of error severity is:

- TM or RM failure: [TX_FAIL]
- Mixed heuristic outcome: [TX_MIXED]
- Heuristic hazard: [TX_HAZARD]
- Possibly-recoverable RM failure: [TX_ERROR]
- Warnings: [TX_OUTSIDE], [TX_ROLLBACK], [TX_COMMitted]
- Success: [TX_OK]

The [TX_EINVAL] and [TX_PROTOCOL_ERROR] codes do not appear in the preceding list because they involve only the interface between AP and TM. The TM generates these results without calling any RMs. For [TX_MIXED], [TX_HAZARD], and [TX_FAIL], more information about the exact nature of the error shall be obtained in a vendor-specified manner.

The following tables describe only those TX calls for which mappings cannot be derived from the hierarchy of error severity.

### Return Codes from xa_open()

The TM opens all RMs when the AP calls `tx_open()`.

<table>
<thead>
<tr>
<th><code>xa_open()</code></th>
<th><code>tx_open()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[XAER_RMERR]</td>
<td>[TX_ERROR]</td>
</tr>
<tr>
<td>[XAER_INVAL]</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAERPROTO]</td>
<td>[TX_FAIL]</td>
</tr>
</tbody>
</table>

Notes:

1. The TM is free to return [TX_OK] in the case where one or more of the RMs returns [XA_OK].

### Return Codes from xa_prepare()

The TM prepares the global transaction at the RM when the AP calls `tx_commit()`.

<table>
<thead>
<tr>
<th><code>xa_prepare()</code></th>
<th><code>tx_commit()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[XA_RDONLY]</td>
<td>[TX_OK]</td>
</tr>
<tr>
<td>[XA_RB*]</td>
<td>[TX_ROLLBACK]</td>
</tr>
<tr>
<td>[XAER_NOTA]</td>
<td>[TX_ROLLBACK]</td>
</tr>
<tr>
<td>[XAER_RMERR]</td>
<td>[TX_ROLLBACK]</td>
</tr>
<tr>
<td>[XAER_RMFAIL]</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAER_INVAL]</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAERPROTO]</td>
<td>[TX_ROLLBACK]</td>
</tr>
</tbody>
</table>

Notes:

1. The XA_RDONLY result ends that RM’s participation in the commitment protocol. If all RMs return XA_RDONLY, the TM returns [TX_OK]. Otherwise, the TM returns a code based on the full commitment protocol at other RMs.

2. These return codes indicate to the TM that the RM in question rolled back work it did on behalf of a global transaction.
3. A return code of [XAER_RMERR] or [XAER_PROTO] from `xa_prepare()` makes no assertion about whether the RM successfully prepared its work. The TM should call all RMs with `xa_rollback()` to attempt to roll back the transaction. If this is unsuccessful, the TM could return [TX_HAZARD] or [TX_FAIL].

**Combined Outcomes During Commit and Rollback**

When two RMs involved in the transaction report different outcomes during a call to `tx_commit()` or `tx_rollback()`, the TM maps the XA result from each RM to a potential TX result using the table below, and then returns the most severe TX result to the AP (see the start of Section B.5 on page 67 for the suggested severity of TX errors). Information in the preceding sections, concerning the treatment of individual XA results from a single RM, remains relevant here.

When there are three or more RMs, the TX return code is given by the most severe error derived from the table.

The following table describes combined outcomes from `xa_commit()` when issued during the second phase of a two-phase commit. The headings RM1 and RM2 do not imply any particular ordering.

<table>
<thead>
<tr>
<th>RM1</th>
<th>RM2</th>
<th>Result to AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[XAER_RMFAIL]</td>
<td>any</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAER_INVAIL]</td>
<td>any</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAER_PROTO]</td>
<td>any</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAER_NOTA]</td>
<td>any</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XA_HEURMIX]</td>
<td>any</td>
<td>[TX_MIXED]</td>
</tr>
<tr>
<td>[XA_HEURHAZ]</td>
<td>any</td>
<td>[TX_HAZARD]</td>
</tr>
<tr>
<td>[XA_RETRY]</td>
<td>any</td>
<td>1</td>
</tr>
<tr>
<td>any commit †</td>
<td>any commit †</td>
<td>[TX_OK]</td>
</tr>
<tr>
<td>any commit †</td>
<td>any rollback ‡</td>
<td>[TX_MIXED]</td>
</tr>
<tr>
<td>any rollback ‡</td>
<td>any rollback ‡</td>
<td>[TX_ROLLBACK]</td>
</tr>
</tbody>
</table>

**Notes:**

1. The [XA_RETRY] code tells the TM to reissue the call. The result the TM returns to the AP depends on the ultimate outcome of the reissued call. If the RM keeps returning [XA_RETRY], the TM may return [TX_HAZARD], or assume it cannot use the RM and return [TX_ERROR] or [TX_FAIL] (see *Origin of Error* on page 62).

2. If [XAER_PROTO] or [XAER_NOTA] is returned on `xa_commit()`, this indicates a serious synchronisation failure between the TM and the RM.

If the *transaction_control* characteristic is set to TX_CHAINED, `xa_start()` may fail after `xa_commit()` is complete. In that case [TX_NO_BEGIN], [TX_ROLLBACK_NO_BEGIN], [TX_MIXED_NO_BEGIN] or [TX_HAZARD_NO_BEGIN] is returned depending on the case. These values correspond to [TX_OK], [TX_ROLLBACK], [TX_MIXED] and [TX_HAZARD] respectively, which would have been returned if `xa_start()` had not failed.

---

† Commitment indications comprise [XA_HEURCOM], and [XA_OK] from `xa_commit()`.  
‡ Rollback indications comprise [XA_HEURRB] and [XAER_RMERR].
The following table describes combined outcomes from `xa_rollback()` when issued during the second phase of a two-phase commit, or when issued as the result of a `tx_rollback()`. The headings RM1 and RM2 do not imply any particular ordering.

<table>
<thead>
<tr>
<th>RM1</th>
<th>RM2</th>
<th>Result to AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[XAER_RMFAIL]</td>
<td>any</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAER_INVFAIL]</td>
<td>any</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XAER_PROTO]</td>
<td>any</td>
<td>[TX_FAIL]</td>
</tr>
<tr>
<td>[XA_HEURMIX]</td>
<td>any</td>
<td>[TX_MIXED]</td>
</tr>
<tr>
<td>[XA_HEURHAZ]</td>
<td>any</td>
<td>[TX_HAZARD]</td>
</tr>
<tr>
<td>[XA_RETRY]</td>
<td>any</td>
<td>1</td>
</tr>
<tr>
<td>[XA_HEURCOM]</td>
<td>[XA_HEURCOM]</td>
<td>3</td>
</tr>
<tr>
<td>any rollback †</td>
<td>any rollback †</td>
<td>4</td>
</tr>
</tbody>
</table>

Notes:

1. The [XA_RETRY] code tells the TM to reissue the call. The result the TM returns to the AP depends on the ultimate outcome of the reissued call. If the RM keeps returning [XA_RETRY], the TM may return [TX_HAZARD], or assume it cannot use the RM and return [TX_ERROR] or [TX_FAIL] (see Origin of Error on page 62).

2. If [XAER_NOTA] is returned on `xa_rollback()`, following a successful `xa_prepare()`, this indicates a serious synchronisation failure between the TM and the RM. The TM should return [TX_FAIL].

3. These return codes indicate to the TM that the RM heuristically committed the work done on behalf of the global transaction. If the AP called `tx_commit()`, it receives [TX_OK]; if the AP called `tx_rollback()`, it receives [TX_COMMITTED].

4. If the AP called `tx_commit()`, it receives [TX_ROLLBACK]; if the AP called `tx_rollback()`, it receives [TX_OK].

If the `transaction_control` characteristic is set to TX_CHAINED, `xa_start()` may fail after `xa_rollback()` is complete. In that case [TX_NO_BEGIN], [TX_COMMITTED_NO_BEGIN], [TX_MIXED_NO_BEGIN] or [TX_HAZARD_NO_BEGIN] is returned depending on the case. These values correspond to [TX_OK], [TX_COMMITTED], [TX_MIXED] and [TX_HAZARD] respectively, which would have been returned if `xa_start()` had not failed.

† Rollback indications comprise [XA_HEURRB], [XAER_NOTA], [XAER_RMERR], and [XAER_OK].
### Index

- `<tx.h>` header ......................................................... 15, 55
- access to resources.................................................. 1
- ACID properties....................................................... 7
- atomicity................................................................. 7
- consistency............................................................. 7
- coordination by TM.................................................. 7
- durability................................................................. 7
- isolation................................................................. 7
- responsibility of RM.................................................. 7
- AP............................................................................... 1
  - component .................................................................. 4
- AP-CRM interface..................................................... 5
- AP-RM interface.......................................................... 5
- AP-TM interface.......................................................... 5
- API............................................................................... 1
  - portability..................................................................... 4
- application ................................................................. 1
  - communication.......................................................... 1
  - distribution............................................................... 1
  - portability................................................................. 1
- program......................................................................... 1
- application program.................................................. 4
  - component .................................................................. 4
  - interface to CRM...................................................... 5
  - interface to RM.......................................................... 5
  - interface to TM.......................................................... 5
- sharing resources...................................................... 1
- atomicity....................................................................... 7
- TM............................................................................... 4
- atomicity of commit.................................................... 8
- autonomy of RM......................................................... 8
- awareness ..................................................................... 8
  - lack of between RMs.................................................. 8
- C language .................................................................. 15
- Common Usage C......................................................... 15
- header.......................................................................... 15
- ISO C ............................................................................ 15
- naming conventions..................................................... 15
- reference manual pages............................................... 19
- TX interface overview................................................. 9
- chained transaction.................................................. 12
- characteristic............................................................ 12
  - commit_return.......................................................... 13, 15, 35, 55
  - commit_return in TXCOMMIT...................................... 39
  - commit_return in TXINFORM........................................ 41
  - commit_return in TXSETCOMMITRET........................... 46
  - commit_return in tx_commit( ) .................................... 23
  - commit_return in tx_info( ) ........................................ 25
  - commit_return in tx_set_commit_return( ) .................. 29
  - public information.................................................... 54
  - transaction_control.................................................... 13, 15, 35, 55
  - transaction_control in TXCOMMIT............................... 39-40
  - transaction_control in TXINFORM............................... 41
  - transaction_control in TXROLLBACK............................. 44-45
  - transaction_control in TXSETTRANCTL.................... 49
  - transaction_control in tx_commit( ) .............................. 23-24
  - transaction_control in tx_info( ) .................................. 25
  - transaction_control in tx_rollback( ) ......................... 27-28
  - transaction_control in............................................... 29
  - tx_set_transaction_control( ) .................................... 31
  - transaction_timeout.................................................... 12-13, 15, 35, 55
  - transaction_timeout in TXBEGIN................................. 36
  - transaction_timeout in TXCOMMIT............................... 39
  - transaction_timeout in TXINFORM............................... 41
  - transaction_timeout in TXROLLBACK............................. 44
  - transaction_timeout in TXSETTIMEOUT.......................... 48
  - transaction_timeout in tx_begin( ) ............................. 20
  - transaction_timeout in tx_commit( ) ............................ 23
  - transaction_timeout in tx_info( ) ............................... 25
  - transaction_timeout in............................................... 32
    - tx_set_transaction_timeout( ) .................................. 32
    - TX-CHAINED......................................................... 49
    - TX-COMMIT-COMPLETED......................................... 46
    - TX-COMMIT-DECISION-LOGGED............................... 46
    - TX-UNCHAINED.................................................... 49
    - TX_CHAINED......................................................... 31
    - TX_COMMIT_COMPLETED........................................ 29
    - TX_COMMIT_DECISION_LOGGED............................... 29
    - tx_set_commit_return( ) ......................................... 11, 29, 52
    - tx_set_transaction_control( ) ................................. 12, 31, 52
    - tx_set_transaction_timeout( ) ................................. 12, 32, 52
    - TX_UNCHAINED.................................................... 31
  - COBOL language....................................................... 33
    - reference manual pages........................................... 33
  - X/Open COBOL.................................................... 33
- commit......................................................................... 8
  - atomic....................................................................... 8
  - decision.................................................................... 4
  - early return.................................................................. 11
  - committing transaction............................................. 7
  - commit_return characteristic.................................... 13
  - in `<tx.h>` header...................................................... 15, 55
Index

in TX-INFO-AREA.................................35
in TXCOMMIT........................................39
in TXINFORM.................................41
in TXSETCOMMITRET.........................46
in tx_commit() .................................23
in tx_info().................................25
in tx_set_commit_return()............11, 29, 52
public information.........................54
TX-COMMIT-COMPLETE.....................46
TX-COMMIT-DECISION-LOGGED...........46
TX_COMMIT_COMPLETED.....................29
TX_COMMIT_DECISION_LOGGED...........29
Common Usage C ..............................15
communication protocol...............1
communication resource manager ....1
component........................................4
interface to AP..............................5
interface to OSI-TP.........................6
interface to TM..............................5
subordinate.................................4
superior.................................4
completion of transaction...............7
coordination.................................4
component..............................3
AP........................................1, 4
AP-CRM interface..........................5
AP-RM interface............................5
AP-TM interface............................5
CRM........................................1, 4
CRM-OSI TP interface...................6
failure.........................................4
interchangeability.........................1
interface between........................5
interoperability............................1
RM........................................1, 4
RM-TM interface............................5
TM........................................1, 4
TM-CRM interface........................5
consistency.................................7
consistent effect of decision............7
consistent state............................7
context.........................................8
control.........................................3
thread of .......................................8
CPI-C interface...........................4-5
CRM..........................................1
component.........................................4
interface to AP............................5
interface to OSI-TP.........................5
interface to TM............................5
subordinate.................................4
superior.................................4
CRM-AP interface...........................5
CRM-OSI TP interface...............6
CRM-TM interface........................5
data structure
TX-COMMIT-COMPLETE....................35
TX-RETURN-STATUS.........................34
TXINFO...........................................25
TXINTRO.................................34
tx_info_f..................................15
XID........................................15
xid_t........................................15
data type
TRANSACTION_STATE..................16
database........................................1
DBMS...........................................4
decision to commit.........................4
decision to commit or roll back........7
definition.................................7
DTP model.................................3
transaction properties....................7
demarcation of transaction...............4
distributed transaction processing (DTP)..7
DTP
implications of..............................7
DTP model......................................1, 3
definition.................................3
durability.................................7
error
return code.............................17
failure of component....................19, 33
failure of system component...........7
file access method........................4
file access system.........................1
flow of control.............................3
functional component
AP........................................4
CRM........................................4
RM........................................4
TM........................................4
functional model..........................3
global transaction.........................4, 8
beginning and completing...............11
header
<tx.h>........................................15
heuristics.................................11
hazard decision...........................11
heuristic completion......................11
mixed decision...........................11
implementation requirements...........53
AP........................................53
Distributed Transaction Processing: The TX (Transaction Demarcation) Specification

Index

RM .................................................................53
TM .................................................................54
implications of DTP ........................................7
interchangeability ...........................................1
interface ..........................................................3
AP-CRM .........................................................5
AP-RM ...........................................................5
AP-TM ............................................................2, 5
between components ........................................5
CPI-C ..............................................................4-5
CRM-OSI TP ....................................................6
function ...........................................................5
illustrated .........................................................5
ISAM .............................................................4-5
SQL ...............................................................5
system-level .....................................................1
TM-CRM .........................................................5
TM-RM ...........................................................5
TX .................................................................5
TxRPC .............................................................4-5
XA .................................................................5
XA+ ...............................................................4-5
XAP-TP ..........................................................4, 6
XATMI ............................................................4-5

interface overview
C-Language ..........................................................9
interoperability ................................................1
introduction to TX interface ...............................1
ISAM interface ...............................................4-5
ISO C .............................................................15
isolation ...........................................................7
location-independence of transaction work ..........7
lock on shared resource .....................................8
mapping (suggested) ..........................................59
function calls ..................................................60
return codes ..................................................62
return codes for a single RM .............................64
return codes for multiple RMs .........................67
TX to XA .........................................................59
XA to TX .........................................................59
method of referencing transaction ......................7
modèle ...............................................................1, 3
functional ........................................................3
modifying shared resource ..................................7
name space
TX .................................................................15
naming conventions .........................................15
native interface ...............................................5
constraints ......................................................5
operations known within RM ............................8
OSI TP standards .............................................4, 6
OSI TP-CRM interface ........................................6
overview of C language interface .........................9

paradigm
CPI-C interface ...............................................4-5
TxRPC interface ...............................................4-5
XATMI interface .............................................4-5

portability
process ................................................................8
protocol ...........................................................1
public information .............................................54

recovery
TM .................................................................4

reference manual pages
C language .......................................................19
COBOL language .............................................33

referencing transaction
method of .......................................................7
resource ..........................................................1
access to ..........................................................1
database ...........................................................1
file access system .............................................1
manager ...........................................................1
system ............................................................8

resource manager
ACID properties responsibility .........................7
interface to AP ..................................................5
interface to TM ..................................................5

return code ......................................................17
error ...............................................................17
success ...........................................................17

return code (C language)
TX_COMMITTED in tx_rollback() .......................27
TX_COMMITTED_NO_BEGIN in .......................28
tx_rollback() ...................................................28
TX EINVAL in tx_set_commit_return() ............29
TX EINVAL in tx_set_transaction_control() ....31
TX EINVAL in tx_set_transaction_timeout() ......32
TX_ERROR in tx_begin() ...................................20
TX_ERROR in tx_close() ....................................22
TX_ERROR in tx_open() .....................................26
TX_FAIL in tx_begin() .....................................20
TX_FAIL in tx_close() .....................................22
TX_FAIL in tx_commit() ...................................24
TX_FAIL in tx_info() .......................................25
TX_FAIL in tx_open() .......................................26
TX_FAIL in tx_rollback() .................................28
TX FAIL in tx_set_commit_return() ...............30
TX FAIL in tx_set_transaction_control() ......31
TX FAIL in tx_set_transaction_timeout() .......32
TX HAZARD in tx_commit() ...............................23
TX HAZARD in tx_rollback() .............................27
TX_HAZARD_NO_BEGIN in tx_commit( )...24
TX_HAZARD_NO_BEGIN in tx_rollback()...27
TX_MIXED in tx_commit()..........................23
TX_MIXED in tx_rollback()........................27
TX_MIXED_NO_BEGIN in tx_commit()............23
TX_MIXED_NO_BEGIN in tx_rollback().........27
TX_NOT_SUPPORTED in...

tx_set_commit_return( )..........................29
TX_NO_BEGIN in tx_commit().....................23
TX_NO_BEGIN in tx_rollback()...............27
TX_OK in tx_begin().............................20
TX_OK in tx_close()..............................22
TX_OK in tx_commit()..........................23
TX_OK in tx_open()..............................26
TX_OK in tx_rollback()........................27
TX_OK in tx_set_commit_return( )...........29
TX_OK in tx_set_transaction_control()......31
TX_OK in tx_set_transaction_timeout().....32
TX_OUTSIDE in tx_begin().....................20
TX_PROTOCOL_ERROR in tx_begin()...........20
TX_PROTOCOL_ERROR in tx_close()...........22
TX_PROTOCOL_ERROR in tx_commit()..........24
TX_PROTOCOL_ERROR in tx_info().............25
TX_PROTOCOL_ERROR in tx_rollback().......28
TX_PROTOCOL_ERROR in...
tx_set_commit_return( )......................29
TX_PROTOCOL_ERROR in...
tx_set_transaction_control()..............31
TX_PROTOCOL_ERROR in...
tx_set_transaction_timeout()..............32
TX_ROLLBACK in tx_commit()...............23
TX_ROLLBACK_NO_BEGIN in...
tx_commit()........................................23
return status (COBOL language)
TX-COMMITTED in TXROLLBACK...............45
TX-COMMITTED-NO-BEGIN in...
TXROLLBACK........................................45
TX-EINVAL in TXSETCOMMITRET...............47
TX-EINVAL in TXSETTIMEOUT..................48
TX-EINVAL in TXSETTRANCTL.................49
TX-ERROR in TXBEGIN..........................36
TX-ERROR in TXCLOSE..........................38
TX-ERROR in TXOPEN...........................43
TX-FAIL in TXBEGIN............................36
TX-FAIL in TXCLOSE............................38
TX-FAIL in TXCOMMIT.........................40
TX-FAIL in TXINFORM...........................41
TX-FAIL in TXOPEN.............................43
TX-FAIL in TXROLLBACK.......................45
TX-FAIL in TXSETCOMMITRET..................47
TX-FAIL in TXSETTIMEOUT....................48
TX-FAIL in TXSETTRANCTL...................49
TX-HAZARD in TXCOMMIT.......................40
TX-HAZARD in TXROLLBACK......................44
TX-HAZARD-NO-BEGIN in TXCOMMIT..........40
TX-HAZARD-NO-BEGIN in TXROLLBACK.........44
TX-MIXED in TXCOMMIT........................39
TX-MIXED in TXROLLBACK.......................44
TX-MIXED-NO-BEGIN in TXCOMMIT..........39
TX-MIXED-NO-BEGIN in TXROLLBACK.........44
TX-NO-BEGIN in TXCOMMIT....................39
TX-NO-BEGIN in TXROLLBACK..................44
TX-NO-SUPPORTED in...
TXSETCOMMITRET...............................46
TX-OK in TXBEGIN..............................36
TX-OK in TXCLOSE..............................38
TX-OK in TXCOMMIT............................39
TX-OK in TXINFORM.............................41
TX-OK in TXOPEN...............................43
TX-OK in TXROLLBACK........................44
TX-OK in TXSETCOMMITRET....................46
TX-OK in TXSETTIMEOUT.......................48
TX-OK in TXSETTRANCTL.....................49
TX-PROTOCOL-ERROR in...
TX_ROLLBACK in TXCOMMIT...............39
TX_ROLLBACK-NO-BEGIN in
TX-PROTOCOL-ERROR in TXROLLBACK...45
TX-PROTOCOL-ERROR in...
TXSETCOMMITRET...............................47
TX-PROTOCOL-ERROR in...
TXSETTRANCTL.................................49
TX-ROLLBACK in TXCOMMIT...................39
TX-ROLLBACK-NO-BEGIN in...
TXCOMMIT.........................................39
RM.................................................1
ACID properties responsibility..............7
component........................................4
opening and closing..........................10
work done across RMs........................7
RM-AP interface...............................5
RM-TM interface...............................5
rolling back transaction....................7
shared resource...............................8
modifying........................................7
permanence of changes to....................7
RM.................................................4
simultaneous updates across RMsin.......8
Index

spanning RMs
  distributed transaction .................................................................7
specification
  CPI-C interface ............................................................5
  TX interface ..................................................................5
  TxRPC interface ..........................................................5
  XA interface ..................................................................5
  XA+ interface ..................................................................5
  XAP-TP interface ..........................................................6
  XATMI interface ...........................................................5

SQL
  interface ...........................................................................5

standard
  ISO C ...........................................................................15
  OSI TP ...........................................................................4, 6

state table .............................................................................51

status of work done anywhere ......................................................7
system component
  failure of ...........................................................................7
  system-level interface .........................................................1
  thread of control ..................................................................8
    same across calls ..............................................................8
  thread state ........................................................................15
  TM ...................................................................................1, 4
    ACID properties coordination ............................................7
    API ..............................................................................5
    atomicity .........................................................................4
    recovery ...........................................................................4
  TM-AP interface ...............................................................5
  TM-CRM interface ............................................................5
  TM-RM interface ...............................................................5

transaction
  actions ...............................................................................1
  beginning and completing .....................................................11
  boundary ...........................................................................4
  branch identifier ..................................................................15
  chained and unchained ..........................................................12
  characteristics ...................................................................13
  commit decision ...................................................................4
  committing ..........................................................................7
  completion ...........................................................................1, 4
  context ................................................................................8
  defining boundaries ..............................................................1
  definition of ..........................................................................1
  demarcation ...........................................................................4-5
  failure ...................................................................................1
  global .................................................................................1, 4, 8
  identifier .............................................................................15
  identifier assigning .............................................................1
  information ...........................................................................12, 15
  manager ................................................................................1

properties ..................................................................................7
recovery ....................................................................................1
RM-internal ...........................................................................8
rolling back .............................................................................7
state information ......................................................................12

transaction control
  tx_begin( ) ...........................................................................5

transaction manager
  ACID properties coordination ..................................................7
  API .......................................................................................5
  atomicity .................................................................................4
  interface to AP .......................................................................5
  interface to CRM .................................................................5
  interface to RM .......................................................................5
  recovery ...................................................................................4
  transaction mode setting ..........................................................35
  transaction timeout ...............................................................12

transaction work
  location-independence of .........................................................7
  transaction_control characteristic ...........................................13
    in <tx.h> header ................................................................15, 55
    in TXINFO-AREA ...............................................................35
    in TXCOMMIT ..................................................................39-40
    in TX Rollsback ................................................................41
    in TXRollenback .............................................................44-45
    in TXSETTRANCTL ..........................................................49
    in tx_commit( ) ...................................................................23-24
    in tx_info( ) .........................................................................25
    in tx_rollback() ...................................................................27-28
    in tx_set_transaction_control() ...........................................12, 31, 52
    TX-CHAINED .....................................................................39, 44, 49
    TX-UNCHAINED ..................................................................39, 44, 49
    TX-CHAINED ......................................................................23, 27, 31, 51
    TX-UNCHAINED ....................................................................23, 27, 31, 51
  TRANSACTION_STATE ..........................................................16
  transaction_timeout characteristic ..........................................13, 32
    in <tx.h> header ................................................................15, 55
    in TXINFO-AREA ...............................................................35
    in TXBEGIN .......................................................................36
    in TXCOMMIT ..................................................................39
    in TX Rollsback ................................................................41
    in TXRollenback .............................................................44
    in TXSETTIMEOUT .............................................................48
    in tx_begin( ) .......................................................................20
    in tx_commit( ) .....................................................................23
    in tx_info( ) .........................................................................25
    in tx_rollback() ...................................................................27
    in tx_set_transaction_timeout() ............................................12, 32, 52

TX interface ..............................................................................5

<tx.h> header ............................................................................15, 55

C language manual pages ..........................................................19

Distributed Transaction Processing: The TX (Transaction Demarcation) Specification 75
Index

chained mode ..............................................12
early return .............................................11
timeout ...................................................12
TX_COMMITTED
  in tx_rollback() ......................................27
TX_COMMITTED_NO_BEGIN ............................17
  in tx_rollback() ......................................28
TX_COMMITTED_COMPLETED
  in tx_set_commit_return() .......................29
TX_COMMIT_DECISION_LOGGED
  in tx_set_commit_return() .......................29
TX_EINVAL ................................................17
  in tx_set_commit_return() .......................29
  in tx_set_transaction_control() ...............31
  in tx_set_transaction_timeout() ...............32
TX_ERROR ................................................17
  in tx_begin() .........................................20
  in tx_close() .........................................22
  in tx_open() ..........................................26
TX_FAIL ..................................................17, 19
  in tx_begin() .........................................20
  in tx_close() .........................................22
  in tx_commit() .......................................24
  in tx_info() ..........................................25
  in tx_open() ..........................................26
  in tx_rollback() ....................................28
  in tx_set_commit_return() ......................30
  in tx_set_transaction_control() ...............31
  in tx_set_transaction_timeout() ...............32
TX_HAZARD .............................................17
  in tx_commit() .......................................23
  in tx_rollback() ....................................27
TX_HAZARD_NO_BEGIN ...............................17
  in tx_commit() .......................................24
  in tx_rollback() ....................................27
tx_info() ...............................................25
tx_info_t data structure ............................15
TX_MIXED ................................................17
  in tx_commit() .......................................23
  in tx_rollback() ....................................27
TX_MIXED_NO_BEGIN .................................17
  in tx_commit() .......................................23
  in tx_rollback() ....................................27
TX_NOT_SUPPORTED ....................................17
  in tx_set_commit_return() .......................29
TX_NO_BEGIN ............................................17
  in tx_commit() .......................................23
  in tx_rollback() ....................................27
TX_OK ....................................................17
  in tx_begin() .........................................20
  in tx_close() .........................................22
in tx_commit() ..........................................23
in tx_open() ............................................26
in tx_rollback() .......................................27
in tx_set_commit_return() .......................29
in tx_set_transaction_control() ...............31
in tx_set_transaction_timeout() ...............32
in tx_open() ............................................10, 26
TX_OUTSIDE .............................................17
  in tx_begin() .........................................20
TX_PROTOCOL_ERROR ...................................17
  in tx_begin() .........................................20
  in tx_close() .........................................22
  in tx_commit() .......................................24
  in tx_info() ..........................................25
  in tx_rollback() ....................................28
  in tx_set_commit_return() .......................29
  in tx_set_transaction_control() ...............31
  in tx_set_transaction_timeout() ...............32
TX_ROLLBACK ...........................................17
  in tx_commit() .......................................23
  in tx_rollback() ....................................11, 27
  chained mode ........................................12
  timeout ...............................................12
TX_ROLLBACK_NO_BEGIN .............................17
  in tx_commit() .......................................23
  in tx_set_commit_return() .......................29
  early return ........................................11
tx_set_transaction_control() .....................31
tx_set_transaction_timeout() .....................32
TX_UNCHAINED .........................................17
  in tx_set_transaction_control() ...............31
  unchained transaction .............................12
undoing work .........................................7
uniform effect of decision ........................7
unit of work ..........................................7
work done .............................................7
work done across RMs ...............................7
work done anywhere .................................7
status of ..............................................7
X/Open publications ................................1
X/Open specification .................................2
  COBOL interface .....................................33
  ISAM interface ......................................5
  SQL interface .......................................5
  TX interface ........................................5
  XA interface ........................................5
  XA+ interface .......................................5
  XAP-TP interface ...................................6
  X/Open-compliant interface ....................7
  XA interface ........................................5
  XA+ interface .......................................4-5

Distributed Transaction Processing: The TX (Transaction Demaration) Specification
XAP-TP interface ....................................................4, 6
XATMI interface ......................................................4-5
XID........................................................................15, 35
XID data structure ....................................................15
xid_t data structure ..................................................15