



IBM Systems - iSeries
UNIX-Type -- XA APIs

Version 5 Release 4





IBM Systems - iSeries
UNIX-Type -- XA APIs

Version 5 Release 4

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 45.

Sixth Edition (February 2006)

This edition applies to version 5, release 4, modification 0 of IBM i5/OS (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 1998, 2006. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

XA APIs	1
Restrictions	2
Supported Database Interfaces	2
Considerations for Remote Databases	3
SQL Server Mode.	3
User Profiles	3
Autocommit	4
APIs	4
XA APIs for Transaction Scoped Locks.	4
xa_close()— Close an XA Resource Manager (Transaction Scoped Locks)	7
Parameters	7
Authorities	7
Return Value	8
Error Messages	8
Related Information	8
Example	8
xa_commit()— Commit an XA Transaction Branch (Transaction Scoped Locks)	9
Parameters	9
Authorities	9
Return Value	9
Error Messages	10
Related Information	11
Example	11
xa_complete()—Test Completion of Asynchronous XA Request (Transaction Scoped Locks)	11
Parameters	11
Authorities	12
Return Value	12
Error Messages	12
Related Information	12
xa_end()—End Work on an XA Transaction Branch (Transaction Scoped Locks)	12
Parameters	13
Authorities	13
Return Value	13
Error Messages	14
Related Information	14
Example	14
xa_forget()— Forget an XA Transaction Branch (Transaction Scoped Locks)	15
Parameters	15
Authorities	15
Return Value	15
Error Messages	16
Related Information	16
Example	16
xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)	16
Parameters	17
Authorities	17
Return Value	17
Error Messages	17
Usage Notes	17
Related Information	20
Example	20
xa_prepare()— Prepare to Commit an XA Transaction Branch (Transaction Scoped Locks)	21
Parameters	21
Authorities	21
Return Value	21
Error Messages	22
Related Information	23
Example	23
xa_recover()— Recover XA Transaction Branches (Transaction Scoped Locks)	23
Parameters	23
Authorities	24
Return Value	24
Error Messages	24
Related Information	24
Example	24
xa_rollback()— Roll Back an XA Transaction Branch (Transaction Scoped Locks)	25
Parameters	25
Authorities	25
Return Value	25
Error Messages	26
Related Information	27
Example	27
xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)	27
Parameters	27
Authorities	28
Return Value	28
Error Messages	29
Related Information	29
Example	29
xa_start_2()—Start an XA Transaction Branch, Extended Version (Transaction Scoped Locks)	29
Example	30
XA APIs for Job Scoped Locks	30
Notes	32
Restrictions for XA APIs for Job Scoped Locks.	32
db2xa_close()—Close an XA Resource Manager (Job Scoped Locks)	34
Example	35
db2xa_commit()—Commit an XA Transaction Branch (Job Scoped Locks)	35
Example	35
db2xa_complete()—Test Completion of Asynchronous XA Request (Job Scoped Locks).	36
db2xa_end()—End Work on an XA Transaction Branch (Job Scoped Locks)	36
Example	37
db2xa_forget()—Forget an XA Transaction Branch (Job Scoped Locks)	37
Example	37
db2xa_open()—Open an XA Resource Manager (Job Scoped Locks)	38
Authorities	38

Usage Notes	38
SQLHOLD Values	39
Example	41
db2xa_prepare()—Prepare to Commit an XA Transaction Branch (Job Scoped Locks)	41
Example	42
db2xa_recover()—Recover XA Transaction Branches (Job Scoped Locks)	42
Example	42
db2xa_rollback()—Roll Back an XA Transaction Branch (Job Scoped Locks)	43

Example	43
db2xa_start()—Start an XA Transaction Branch (Job Scoped Locks)	43

Appendix. Notices 45

Programming Interface Information	46
Trademarks	47
Terms and Conditions	48

XA APIs

DB2^(R) UDB for iSeries^(TM) provides two sets of XA APIs:

- “XA APIs for Transaction Scoped Locks” on page 4
- “XA APIs for Job Scoped Locks” on page 30

Before you use the XA APIs, you should read the following publications, which describe the X/Open Distributed Transaction Processing model in detail.

- X/Open Guide, February 1996, Distributed Transaction Processing: Reference Model, Version 3 (ISBN:1-85912-170-5, G504), The Open Group.
- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

The model consists of five basic components:

- An application program, which defines transaction boundaries and specifies actions that constitute a transaction.
- Resource managers, such as databases or file access systems, which provide access to resources.
- A transaction manager, which assigns identifiers to transactions, monitors their progress, and takes responsibility for transaction completion and for coordinating failure recovery.
- Communications resource managers, which control communications between distributed applications within or across transaction manager domains.
- A communications protocol, which provides the underlying communications between distributed applications. The protocol is supported by protected resource managers.

This section explains the use of DB2 UDB for iSeries as an X/Open-compliant resource manager, and therefore is concerned only with the first three components of this model. More specifically, it documents the XA interface, which is the portion of the XA Distributed Transaction Processing model that transaction managers and resource managers use to communicate. The XA interface is a bidirectional interface, which consists of a set of UNIX^(R)-type APIs.

The XA specification requires the resource manager to provide a **switch** that gives the transaction manager access to these APIs. The switch allows an administrator to change the set of resource managers that are linked with a program without having to recompile the application. This switch is a data structure that contains the resource manager’s name, non-null pointers to the resource manager’s APIs, a flag, and a version word.

DB2 UDB for iSeries provides a switch for each set of XA APIs. Each switch is exported by the QTNXADTP service program. The switch for the “XA APIs for Transaction Scoped Locks” on page 4 is called *xa_switch*. The switch for the “XA APIs for Job Scoped Locks” on page 30 is called *db2xa_switch*. The flags in each switch provide information about the resource manager including the facts that migration of associations is not supported and asynchronous requests are not allowed. They also contain an array of procedure pointers that give addressability to the XA APIs. The XA APIs are typically called by a transaction manager using these pointers rather than by name. This precludes the transaction manager from having to know the actual function names and from having to link to the service program that actually contains the functions.

The XA specification requires each resource manager to provide a header file that defines data structures and constants common to the operation of transaction managers and resource managers. The DB2 UDB

for iSeries XA resource manager ships two header files in file H, library QSYSINC. Member XA contains a header file that is compatible with the XA architecture. Member QTNXADTP contains a header file that is not compatible with the XA architecture. Some of the structure and variable names in header file QTNXADTP have the prefix "db2." Either file can be used, but it is recommended that the XA header file be used rather than the QTNXADTP header file. The examples at the end of the XA APIs assume you use the XA header file.


See Commitment Control for additional information on commitment control and XA transactions.

Restrictions

Supported Database Interfaces

X/Open applications must use SQL interfaces to access resources managed by DB2 UDB for iSeries. Both the embedded and call level interface (CLI) SQL interfaces are supported.


In particular, the following interfaces are not supported:


-  Traditional system interfaces for updating local database files or distributed data management (DDM) files, such as Control language (CL) or high-level languages (HLL), except that the following cases are allowed:
 - Triggers, stored procedures and user defined functions may use these interfaces for updating local files.
 - The application may use these interfaces directly for updating DDM files if all the following are true:
 - The "XA APIs for Transaction Scoped Locks" on page 4 are used.
 - i5/OS^(TM) thread is specified as the XA thread of control via the xainfo string of the "xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)" on page 16 API.
 - SQL server mode is *not* used.



- The Process Extended Dynamic SQL (QSQPRCED) API.
- The Query (QQQQRY) API.
- The commitment control API interfaces documented in the Journal and Commit APIs category.

 The DB2 UDB for iSeries XA transaction support may also be driven indirectly via these interfaces:

- IBM DB2 UDB for clients other than iSeries. Refer to *Designing for XA compliant transaction managers* in Administration Guide: Planning. 
- The iSeries Access ODBC driver, which includes support of the Microsoft Transaction Server (MTS). Refer to Distributed transaction support.
- The iSeries Extended Dynamic Remote SQL (EDRS) APIs. Refer to Extended Dynamic Remote SQL (EDRS) APIs.

If you are writing a new application, you should use the "XA APIs for Transaction Scoped Locks" on page 4. These APIs have fewer restrictions than the "XA APIs for Job Scoped Locks" on page 30, and provide better performance in the following situations: 

- If multiple SQL connections are ever used to work on a single XA transaction branch.
- If a single SQL connection is used to work on multiple, concurrent XA transaction branches.

In these situations, a separate job must be started to run XA transaction branches when the "XA APIs for Job Scoped Locks" on page 30 are used.



Considerations for Remote Databases

Local relational databases may be used for both the “XA APIs for Transaction Scoped Locks” on page 4 or the “XA APIs for Job Scoped Locks” on page 30 without regard to communications protocols. Local databases include those defined for an Independent ASP.

When using the “XA APIs for Job Scoped Locks” on page 30 for a remote relational database, the RDB connection method must be Distributed Unit of Work (*DUW), and the remote location may be defined for either TCP/IP or SNA LU6.2 connections.

When using the “XA APIs for Transaction Scoped Locks” on page 4 for a remote relational database, the RDB connection method must be Distributed Unit of Work (*DUW), the remote location must be defined for TCP/IP connections, and the remote system must support XAMGR Level 7 of the DRDA architecture. When using these APIs, it is also important to understand a difference in connection behavior between local and remote databases. See the description of the THDCTL keyword in the “xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)” on page 16 xainfo string for details. <<

SQL Server Mode

Transactions that require the use of an XA resource manager must be performed in SQL server jobs >> if you are using the “XA APIs for Job Scoped Locks” on page 30, or if you are using the “XA APIs for Transaction Scoped Locks” on page 4 against a local database. If you are using the “XA APIs for Transaction Scoped Locks” on page 4 against a remote database, use of SQL server jobs is required when using the SQL connection as the XA thread of control, and optional when using the i5/OS thread as the XA thread of control. << An SQL server job is a job whose server mode for Structured Query Language attribute has been set to *YES. Use the Change Job (QWTCHGJOB) API to control the setting of this attribute. The “xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)” on page 16 and “db2xa_open()—Open an XA Resource Manager (Job Scoped Locks)” on page 38 APIs will set the server mode attribute to *YES if the attribute has not already been set, >> except when the “xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)” on page 16 API is used for a remote database and the i5/OS thread is specified as the XA thread of control. See the description of the THDCTL keyword in the “xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)” on page 16 xainfo string for more information regarding the XA thread of control. << For additional information about SQL server jobs, see DB2 UDB for iSeries SQL Programming Concepts in the Information Center and the question on What is CLI Server Mode? in the DB2 Universal Database for iSeries SQL CLI Frequently Asked

Questions. 

User Profiles

It is expected that most transaction managers will use the same user profile for all SQL connections. If the “xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)” on page 16 or “db2xa_open()—Open an XA Resource Manager (Job Scoped Locks)” on page 38 APIs are used before the connections are started, this can be accomplished by specifying the same user profile for the *xainfo parameter of each “xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)” on page 16 or “db2xa_open()—Open an XA Resource Manager (Job Scoped Locks)” on page 38 API call. XA applications generally do not use the resource manager’s native security mechanisms to limit access to data. Rather, this is done at the application or transaction manager level.

>>

Autocommit

Autocommit must not be in effect for any connection that is used to perform work during a XA transaction branch, regardless of whether the connection was established on the iSeries or some client that drives the iSeries XA transaction support. The default setting for the autocommit attribute is FALSE for connections established on the iSeries. Some clients may implicitly change the autocommit attribute for the connection to an iSeries server when a XA transaction branch is started. Refer to the documentation for each client for information about its autocommit behavior. <<

Top | UNIX-Type APIs | APIs by category

APIs

These are the APIs for this category.

XA APIs for Transaction Scoped Locks

The following XA APIs for Transaction Scoped Locks are provided by the DB2^(R) UDB for iSeries^(TM) XA resource manager for use by a transaction manager:

- “xa_close()— Close an XA Resource Manager (Transaction Scoped Locks)” on page 7 (Close an XA Resource Manager (Transaction Scoped Locks)) closes a currently open resource manager in the thread of control.
- “xa_commit()— Commit an XA Transaction Branch (Transaction Scoped Locks)” on page 9 (Commit an XA Transaction Branch (Transaction Scoped Locks)) commits the work associated with **xid*.
- “xa_complete()—Test Completion of Asynchronous XA Request (Transaction Scoped Locks)” on page 11 (Test Completion of Asynchronous XA Request) waits for the completion of an asynchronous operation.
- “xa_end()—End Work on an XA Transaction Branch (Transaction Scoped Locks)” on page 12 (End Work on an XA Transaction Branch (Transaction Scoped Locks)) is called when when an application thread of control finishes or needs to suspend work on a transaction branch.
- “xa_forget()— Forget an XA Transaction Branch (Transaction Scoped Locks)” on page 15 (Forget an XA Transaction Branch (Transaction Scoped Locks)) is called to forget about a heuristically completed transaction branch.
- “xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)” on page 16 (Open an XA Resource Manager (Transaction Scoped Locks)) is called to open the XA resource manager and to prepare it for use in the XA distributed transaction environment.
- “xa_prepare()— Prepare to Commit an XA Transaction Branch (Transaction Scoped Locks)” on page 21 (Prepare to Commit an XA Transaction Branch (Transaction Scoped Locks)) is called to request that a resource manager prepare for commitment any work performed on behalf of **xid*.
- “xa_recover()— Recover XA Transaction Branches (Transaction Scoped Locks)” on page 23 (Recover XA Transaction Branches (Transaction Scoped Locks)) is called during recovery to obtain a list of transaction branches that are currently in a prepared or heuristically completed state.
- “xa_rollback()— Roll Back an XA Transaction Branch (Transaction Scoped Locks)” on page 25 (Roll Back an XA Transaction Branch (Transaction Scoped Locks)) is called to roll back work performed on behalf of the transaction branch.
- “xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 (Start an XA Transaction Branch (Transaction Scoped Locks)) informs a resource manager that an application may do work on behalf of a transaction branch.
- “xa_start_2()—Start an XA Transaction Branch, Extended Version (Transaction Scoped Locks)” on page 29 (Start an XA Transaction Branch, Extended Version (Transaction Scoped Locks)) informs a resource manager that an application may do work on behalf of a transaction branch.

» When using the XA APIs for Transaction Scoped Locks, the XA thread of control is normally considered to be the i5/OS^(TM) thread from which transactional work is requested, regardless of what SQL connection is used to perform that work. However, in some cases, it is useful to use the SQL connection, rather than the i5/OS thread, as the XA thread of control. This can be accomplished by using extensions provided with the DB2 UDB for iSeries implementation of the **SQLSetConnectAttr** API to start and end work on an XA transaction, rather than the “xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 and “xa_end()—End Work on an XA Transaction Branch (Transaction Scoped Locks)” on page 12 APIs. In this environment:

- use **SQLSetConnectAttr** to set attribute `SQL_ATTR_TXN_EXTERNAL` to `SQL_TRUE` to indicate that the CLI connection is to be used as the XA thread of control.
- use **SQLSetConnectAttr** to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_CREATE` to start a transaction branch (similar to “xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 with no flags).
- use **SQLSetConnectAttr** to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_FIND` to join a transaction branch (similar to “xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 with the `TMJOIN` flag).
- use **SQLSetConnectAttr** to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_RESUME` to resume a suspended transaction branch (similar to “xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 with the `TMRESUME` flag).
- use **SQLSetConnectAttr** to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_END` to end work for a transaction branch and close SQL and legacy cursors (similar to “xa_end()—End Work on an XA Transaction Branch (Transaction Scoped Locks)” on page 12 with the `TMSUCCESS` flag).
- use **SQLSetConnectAttr** to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_END_FAIL` to end work for a transaction branch, close SQL and legacy cursors and mark the transaction rollback only (similar to “xa_end()—End Work on an XA Transaction Branch (Transaction Scoped Locks)” on page 12 with the `TMFAIL` flag).
- use **SQLSetConnectAttr** to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_HOLD` to end work on a transaction branch and close SQL cursors except those that were defined `WITH HOLD` and close legacy cursors (no XA API equivalent). When using this option, each transaction branch must be committed or rolled back before using the connection to perform work on another transaction branch. Also, use `SQL_TXN_END` rather than `SQL_TXN_HOLD` for the last transaction branch prior to a disconnect. Unpredictable results may occur if these rules are not followed.
- use **SQLSetConnectAttr** to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_CLEAR` to suspend work on a transaction branch and close legacy cursors (similar to “xa_end()—End Work on an XA Transaction Branch (Transaction Scoped Locks)” on page 12 with the `TMSUSPEND` flag).

Regardless of whether using the i5/OS thread or the SQL connection as the thread of control as far as transactional work is concerned, the “xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)” on page 16 API must always be issued in any i5/OS thread prior to using any of the XA APIs. «

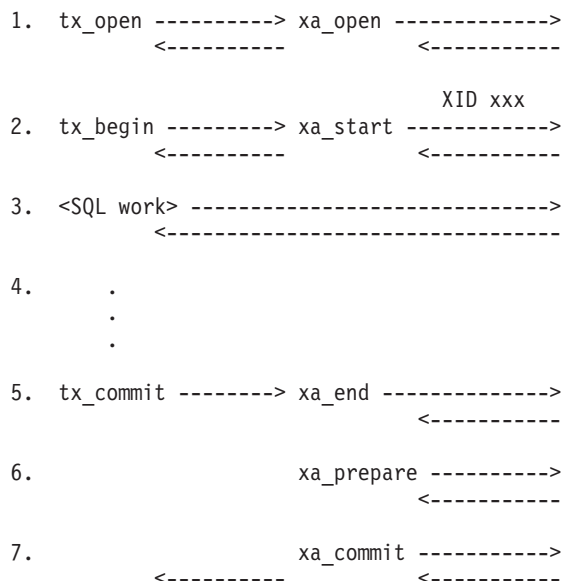
The following example shows the interactions between the application program, transaction manager, and the XA resource manager during a typical transaction branch when the XA APIs for Transaction Scoped Locks are used. The actual interactions that occur during a transaction will vary depending on factors such as the following:

- Whether the transaction is committed or rolled back
- Whether the one- or two-phase commit protocol is used with the XA resource manager
- Whether multiple threads are used to perform the work of a transaction branch

Refer to the X/Open XA Specification for details.

Example Using XA APIs for Transaction Scoped Locks, » Thread of Control is i5/OS Thread «

HLL	XA	XA
Application	Transaction	Resource
Program	Manager	Manager



Notes

1. The application uses the X/Open Transaction Demarcation (TX) **tx_open()** interface to open all the resource managers that are linked with the transaction manager. The transaction manager uses the **xa_open()** interface to open an instance of the XA resource manager. The transaction manager may open multiple XA resource managers that will participate in XA transactions. The transaction manager assigns a resource manager identifier (ID) to each resource manager instance. The resource manager ID uniquely identifies the instance within the thread of control in which the application is running.
2. The application uses the TX **tx_begin()** interface to begin a transaction. For each resource manager that will participate in XA transactions, the transaction manager generates a transaction branch identifier (XID) and uses the XA **xa_start()** interface to start a transaction branch.
3. The application uses SQL interfaces to access resources managed by DB2 UDB for iSeries.
4. The application continues its transaction. It may access other resource managers as appropriate.
5. When the transaction has been completed, the application uses the TX **tx_commit()** interface to commit the work. The transaction manager uses the XA **xa_end()** interface to end the transaction branch.
6. The transaction manager uses the XA **xa_prepare()** interface to prepare the resources for commitment.
7. The transaction manager uses the XA **xa_commit()** interface to commit the resources after all the resource managers involved in the transaction have successfully prepared their resources for commitment. When the commit operation is complete, the application can begin another transaction using the TX **tx_begin()** interface.



Example Using XA APIs for Transaction Scoped Locks, Thread of Control is SQL Connection

1. xa_open (specify THDCTL=C on xainfo string parameter). SQLAllocHandle CLI API to allocate SQL environment handle
2. SQLAllocHandle CLI API to allocate SQL connection handle
3. SQLConnect CLI API to establish connection
4. SQLSetConnectAttr to set attribute SQL_ATTR_TXN_EXTERNAL to SQL_TRUE

5. `SQLSetConnectAttr` to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_CREATE` (this starts a XA transaction branch, similar to `start`)
6. Perform SQL work over the CLI connection
7. `SQLSetConnectAttr` to set attribute `SQL_ATTR_TXN_INFO` to `SQL_TXN_END` (this ends the XA transaction branch, similar to `start`)
8. `xa_prepare`
9. `xa_commit`

Notes

1. As with the XA protocol, once the association with a transaction branch has been ended by setting connection attribute `SQL_ATTR_TXN_INFO` to `SQL_TX_END`, `SQL_TX_CLEAR` or `SQL_TX_HOLD`, the connection may be used to start, join or resume other transaction branches before committing or rolling back the transaction branch whose association was just ended. Also, the XA commit or rollback requests may be issued from any thread in any job. They are not required to be issued from the same thread that issued the `SQLSetConnectAttr` to start the transaction branch.
2. See SQL call level interface for detailed information on the CLI APIs.



Top | UNIX-Type APIs | APIs by category

xa_close()— Close an XA Resource Manager (Transaction Scoped ,Locks)

Syntax

```
#include <xa.h>
```

```
int xa_switch.xa_close_entry(char *xa_info,
                             int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **xa_close()** to close a currently open resource manager in the thread of control. After this call, the resource manager cannot participate in global transactions on behalf of the calling thread until it is reopened.

Parameters

xa_info

(Input) A pointer to a 256-byte, null-terminated character string that contains information used to close the resource manager. No information is currently allowed in this string. It must be a null string or contain only blanks with a null terminator.

rmid (Input) An integer value that the transaction manager generated when calling **xa_open()**. The `rmid` identifies the resource manager.

flags (Input) The following are valid settings of *flags*.

TMNOFLAGS: 0x00000000L Perform the close operation normally.

Authorities

None

Return Value

- 6 [XAER_PROTO]
`xa_close()` was not successful. The function was called in an improper context.
- 5 [XAER_INVALID]
`xa_close()` was not successful. Incorrect arguments were specified.
- 3 [XAER_RMERR]
`xa_close()` was not successful. The resource manager detected an error when it closed the resource.
- 2 [XAER_ASYNC]
`xa_close()` was not successful. The resource manager does not support asynchronous operations.
- 0 [XA_OK]
`xa_close()` was successful.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    char *xa_info;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t xa_switch;

    retcode =
        xa_switch.xa_close_entry(xa_info, rmid, flags);
}
```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_commit()— Commit an XA Transaction Branch (Transaction Scoped Locks)

Syntax

```
#include <xa.h>

int xa_switch.xa_commit_entry(XID *xid,
    int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **xa_commit()** to commit the work associated with **xid*. All changes that were made to resources managed by DB2 UDB for iSeries during the transaction branch are made permanent.

Parameters

- xid** (Input) A pointer to the transaction branch identifier. This identifier was generated by the transaction manager when the transaction branch was started.
- rmid** (Input) An integer value that the transaction manager generated when calling **xa_open()**. The rmid identifies the resource manager.
- flags** (Input) Following are the valid settings of *flags*.
- TMNOWAIT*: 0x10000000L Do not commit the transaction if a blocking condition exists.
 - TMONEPHASE*: 0x40000000L Use the one-phase commit optimization for the specified transaction branch.
 - TMNOFLAGS*: 0x00000000L Use if no other flags are set.

Authorities

None

Return Value

The following values may be returned only if *TMONEPHASE*(0x40000000L) was set in the *flags* parameter.

- 100 [XA_RBROLLBACK]
The transaction branch was rolled back for an unspecified reason.
- 101 [XA_RBCOMMFAIL]
A communications failure occurred within the resource manager.
- 102 [XA_RBDEADLOCK]
A deadlock condition was detected within the resource manager.
- 103 [XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.
- 104 [XA_RBOTHER]
The resource manager rolled back the transaction branch for a reason not on this list.
- 105 [XA_RBPROTO]
A protocol error occurred in the resource manager.

- 106 [XA_RBTIMEOUT]
A timeout occurred in the resource manager.
- 107 [XA_RBTRANSIENT]
A transient error was detected in the resource manager.

The following values may be returned for all *flags* settings.

- 7 [XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.
- 6 [XAER_PROTO]
xa_commit() was not successful. Function was called in an improper context.
- 5 [XAER_INVALID]
xa_commit() was not successful. Incorrect arguments were specified.
- 4 [XAER_NOTA]
The specified *xid* is not known by the resource manager.
- 3 [XAER_RMERR]
xa_commit() was not successful. The resource manager detected an error when committing the transaction branch.
- 2 [XAER_ASYNC]
xa_commit() was not successful. The resource manager does not support asynchronous operations.
- 0 [XA_OK]
xa_commit() was successful.
- 4 [XA_RETRY]
The resource manager is unable to commit the transaction branch at this time. *TMNOWAIT*(0x10000000L) was set and a blocking condition exists. All resources held on behalf of **xid* remain in a prepared state. The transaction manager should issue **xa_commit()** again at a later time.
- 5 [XA_HEURMIX]
Work on the transaction branch was partially committed and partially rolled back.
- 6 [XA_HEURRB]
Work on the transaction branch was heuristically rolled back.
- 7 [XA_HEURCOM]
Work on the transaction branch was heuristically committed.
- 8 [XA_HEURHAZ]
Work on the transaction branch may have been heuristically completed.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t xa_switch;

    retcode =
        xa_switch.xa_commit_entry(xid, rmid, flags);
}
```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_complete()—Test Completion of Asynchronous XA Request (Transaction Scoped Locks)

Syntax

```
#include <xa.h>

int xa_switch.xa_complete_entry(int *handle,
    int *retval, int rmid, long flags)
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **xa_complete()** to wait for the completion of an asynchronous operation. Asynchronous operations are not supported by the DB2 UDB for iSeries resource manager. This function is provided only for compliance with the X/Open XA Specification.

Parameters

handle

(Input) A pointer to an integer value returned by an XA function that had TMASYNC specified.

retval (Output) A pointer to the integer return value of the asynchronous function.

rmid (Input) An integer value that the transaction manager generated when calling **xa_open()**. The rmid identifies the resource manager.

flags (Input) The follow are valid settings of *flags*.

TMMULTIPLE: 0x00400000L Test completion of any outstanding asynchronous operation.

TMNOWAIT: 0x10000000L Test for completion without blocking.

TMNOFLAGS: 0x00000000L Use if no other flags are set.

Authorities

None

Return Value

-6 [XAER_PROTO]

`xa_complete()` was not successful. `TMUSEASYNC 0x00000004L` was not set in the `flags` element of the XA resource manager's `xa_switch_t` structure. Asynchronous operations are not supported.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

`xa_end()`—End Work on an XA Transaction Branch (Transaction Scoped Locks)

Syntax

```
#include <xa.h>
```

```
int xa_switch.xa_end_entry(XID *xid, int rmid,  
    long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls `xa_end()` when an application thread of control finishes or needs to suspend work on a transaction branch. When `xa_end()` successfully returns, the calling thread of control is no longer associated with the transaction branch, but the branch still exists.

If the `TMSUSPEND` flag is not specified, all SQL cursors used while the thread was associated with this transaction branch are closed. Files left open by a procedure, trigger or function that used legacy file access methods are closed regardless of flag settings.

Parameters

- *xid** (Input) A pointer to the transaction branch identifier. This identifier was generated by the transaction manager when the transaction branch was started.
- rmid** (Input) An integer value that the transaction manager generated when calling `xa_open()`. The `rmid` identifies the resource manager.
- flags** (Input) The following are valid settings of *flags*. One, and only one, of `TMSUSPEND`, `TMSUCCESS`, or `TMFAIL` must be set.
- TMSUSPEND*: 0x02000000L Suspend a transaction branch on behalf of the calling thread. The transaction manager must resume or end the suspended association in the current thread.
- TMSUCCESS*: 0x04000000L The portion of work has succeeded.
- TMFAIL*: 0x20000000L The portion of work has failed.

Authorities

None

Return Value

The following return codes indicate that the resource manager has marked the work performed on this transaction branch as rollback-only.

- 100 [XA_RBROLLBACK]
The transaction branch was marked rollback-only for an unspecified reason.
- 101 [XA_RBCOMMFAIL]
A communications failure occurred within the resource manager.
- 102 [XA_RBDEADLOCK]
A deadlock condition was detected within the resource manager.
- 103 [XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.
- 104 [XA_RBOTHER]
The resource manager marked the transaction branch rollback-only for a reason not on this list.
- 105 [XA_RBPROTO]
A protocol error occurred in the resource manager.
- 106 [XA_RBTIMEOUT]
A timeout occurred in the resource manager.
- 107 [XA_RBTRANSIENT]
A transient error was detected by the resource manager.

Other return codes:

- 7 [XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.
- 6 [XAER_PROTO]
Function was called in an improper context.
- 5 [XAER_INVALID]
Incorrect arguments were specified.

- 4 [XAER_NOTA]
The specified **xid* is not known by the resource manager.
- 3 [XAER_RMERR]
xa_end() was not successful. The resource manager detected an error when ending the transaction branch.
- 2 [XAER_ASYNC]
xa_end() was not successful. The resource manager does not support asynchronous operations.
- 0 [XA_OK]
xa_end() was successful.
- 9 [XA_NOMIGRATE]
The resource manager was unable to prepare the transaction context for migration. The resource manager has suspended the association. The transaction manager can resume the association in the current thread only.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t xa_switch;

    retcode =
        xa_switch.xa_end_entry(xid, rmid, flags);
}
```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_forget()— Forget an XA Transaction Branch (Transaction Scoped Locks)

Syntax

```
#include <xa.h>
```

```
int xa_switch.xa_forget_entry(XID *xid,  
    int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **xa_forget()** to forget about a heuristically completed transaction branch. After this call, the **xid* is no longer valid.

Parameters

- *xid** (Input) A pointer to the transaction branch identifier. This identifier was generated by the transaction manager when the transaction branch was started.
- rmid** (Input) An integer value that the transaction manager generated when calling **xa_open()**. The *rmid* identifies the resource manager.
- flags** (Input) The following are valid settings of *flags*.
TMNOFLAGS: 0x00000000L Perform the forget operation normally.

Authorities

None

Return Value

- 7 [XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.
- 6 [XAER_PROTO]
xa_forget() was not successful. Function was called in an improper context.
- 5 [XAER_INVALID]
xa_forget() was not successful. Incorrect arguments were specified.
- 4 [XAER_NOTA]
The specified *xid* is not known by the resource manager.
- 3 [XAER_RMERR]
xa_forget() was not successful. The resource manager detected an error when forgetting the transaction branch.
- 2 [XAER_ASYNC]
xa_forget() was not successful. The resource manager does not support asynchronous operations.
- 0 [TM_OK]
xa_forget() was successful.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
<i>CPE3418 E</i>	Possible APAR condition or hardware failure.
<i>CPF3CF2 E</i>	Error(s) occurred during running of &1 API.
<i>CPF9872 E</i>	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t xa_switch;

    retcode =
        xa_switch.xa_forget_entry(xid, rmid, flags);
}
```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)

Syntax

```
#include <xa.h>

int xa_switch.xa_open_entry(char *xa_info,
    int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **xa_open()** to open the XA resource manager and to prepare it for use in the XA distributed transaction environment. This function must be called before any other resource manager (*xa_**) calls are made.

Parameters

**xa_info*

(Input) A pointer to a null-terminated string that contains information used to initialize the resource manager. See the Usage Notes for details on what this string should contain.

rmid (Input) A number generated by the transaction manager to identify this instance of the XA resource manager. This *resource manager identifier* is passed to the other XA functions to identify which instance of the resource manager for which the function is called.

flags (Input) The following are valid settings of *flags*.

TMNOFLAGS: 0x00000000L Perform the open operation normally.

Authorities

None

Return Value

-6 [XAER_PROTO]

xa_open() was not successful. Function was called in an improper context.

-5 [XAER_INVALID]

xa_open() was not successful. Incorrect arguments were specified.

-3 [XAER_RMERR]

xa_open() was not successful. The resource manager detected an error when opening the resource manager.

-2 [XAER_ASYNC]

xa_open() was not successful. The resource manager does not support asynchronous operations.

0 [TM_OK]

xa_open() was successful.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.


Usage Notes

- A pointer to the *xa_info* character string is passed on the *xa_open()* function. The character string contains information required by the XA resource manager. This information affects the behavior of DB2 UDB for iSeries when running as an XA resource manager. The *xa_info* string is a series of keyword specifications, each of which consists of:

- A keyword.
- The '=' character.
- A keyword value.

For example:


TMNAME=YourTM RDBNAME=SYSABC lockwait=300

- The restrictions on the data in the *xa_info* character string are:
 - There must be no blanks between the keyword and the '=' or between the '=' and the keyword value.
 - The *xa_info* string must neither begin nor end with the '=' character.
 - There must be at least one blank between each keyword specification.
 - Keywords and keyword values, except the PASSWORD keyword value, are not case-sensitive; keyword values on system displays or messages are shown in uppercase. The PASSWORD keyword value is case-sensitive.
 - If the PASSWORD keyword is specified, its value is assumed to be represented in the job default CCSID of the job that calls the *xa_open()* function.
 - The *xa_info* string is limited to 1024 bytes and must be null-terminated. Note that this is longer than the 256 byte maximum architected in the XA Specification, however the longer length is required for iSeries long password support. If a null byte ('00'x) is not found in the first 1024 bytes, [XAER_INVALID] is returned.
 - The *xa_info* string value is treated as character data and is not converted.
 - The return value [XAER_INVALID] will be returned if a keyword is specified that is not documented in *xainfo String Keywords and Values* (page 18).
-  A SQL server job is a job whose server mode for Structured Query Language attribute has been set to *YES. SQL server jobs are required when a local database is specified for the RDBNAME keyword, or when a remote database is specified for the RDBNAME keyword and C is specified for the THDCTL keyword. In these cases, the *xa_open()* API will implicitly set the server mode attribute for the job if it has not already been set. SQL server jobs are optional when a remote database is specified for the RDBNAME keyword and T is specified or defaulted for the THDCTL keyword. In this case, the user may choose not to use SQL server jobs, or may set the server mode attribute using the Change Job (QWTCHGJOB) API. For additional information about SQL server jobs, see DB2 UDB for iSeries SQL Programming Concepts in the Information Center and the question on What is CLI Server Mode? in the DB2 Universal Database for iSeries SQL CLI Frequently Asked Questions.





- When T is specified for the THDCTL keyword, connection behavior is different for remote connections than it is for local connections.


If a remote database is specified for the RDBNAME keyword, a connection to the remote database is opened immediately. This connection is used for all XA requests made in that thread, and all subsequent SQL work requested in that thread, even if the CLI is used to open multiple connection handles. In other words, no more than one physical connection is created for the thread even if the CLI is used to request multiple connections from that thread.

If a local database is specified for the RDBNAME keyword, no connection is immediately established, and each connection request establishes a unique physical connection to the database. 

xainfo String Keywords and Values

Keyword Name	Keyword Value
LOCKWAIT	<p>The maximum number of seconds that the system will wait on any lock request during transaction branches started by this thread. Lock wait time values that are specified by other system interfaces will be used only if they are smaller than this value.</p> <p>If not specified, lock wait time values specified by other system interfaces are used. The maximum value that may be specified is 99999999.</p>

Keyword Name	Keyword Value
PASSWORD	<p>The password to be used in conjunction with the user when accessing the relational database. This value is used only if the USER keyword is also specified. If specified, the password value is assumed to be represented in the job default CCSID of the job that calls the xa_open() API. If the specified password value contains any null bytes ('00'x) or blanks ('40'x), the PWDLEN keyword must also be specified. The length of the password value must not exceed 512 bytes.</p> <p>If this keyword is not specified, PASSWORD defaults to 10 blanks.</p>
PWDLEN	<p>The length, in bytes, of the password. This value must not exceed 512. This keyword must be specified if the value specified for the PASSWORD keyword contains any null bytes ('00'x) or blanks ('40'x). If specified, the keyword must appear before the PASSWORD keyword.</p> <p>If this keyword is not specified, the length of the specified PASSWORD value is determined by the location of the first null byte ('00'x) or blank ('40'x) following the PASSWORD keyword. If the PASSWORD keyword is not specified, the value specified for this keyword is ignored.</p>
RDBNAME	<p>A 1- to 18-character name identifying the relational database that the transaction manager will use for XA transaction branches in this thread. If there is an entry in the relational database directory with Remote Location value *LOCAL, then special value *LOCAL may be used to identify that database.</p> <p>This is a required keyword. If this keyword is not specified, [XAER_INVALID] is returned.</p> <p>Once a thread calls xa_open() with a particular rmid and RDBNAME combination, the rmid may not be used on subsequent xa_open() calls unless the same RDBNAME value is used. Likewise, the RDBNAME value may not be used on subsequent xa_open() calls unless the same rmid is used. If a subsequent call is made with the same rmid and RDBNAME combination, but other values in the xa_info string are different, the values on the first call remain in effect and a CPI836A informational message is sent to the joblog.</p>
 TBLCS	<p>Indicates whether loosely coupled threads of control (those working on transaction branches with the same global transaction identifier (GTRID), but different branch qualifiers (BQUALs)), share locks. Specify value N if they are not to share locks, meaning that resource deadlock may occur between the transaction branches. Specify value S if you want them to share locks, meaning that resource deadlock will not occur between the transaction branches.</p> <p>If S is specified, the last transaction branch to be committed or rolled back will commit or rollback the changes for all the transaction branches with the same GTRID. xa_prepare requests for the other transaction branches will complete those transaction branches and return XA_RDONLY, but changes made and locks acquired for those transaction branches will remain pending until the last transaction branch with the same GTRID is completed.</p> <p>If this keyword is not specified, TBLCS defaults to N. </p>

Keyword Name	Keyword Value
THDCTL	<p>Indicates the XA thread of control. Specify value T to use the i5/OS thread as the XA thread of control. Specify value C to use the SQL connection as the XA thread of control.</p> <p>Value C should be used only when the transaction manager will use the CLI interface to establish connections, and the SQLSetConnectAttr API to start and end XA transaction branches, rather than the “xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 and “xa_end()—End Work on an XA Transaction Branch (Transaction Scoped Locks)” on page 12 APIs. See “XA APIs for Transaction Scoped Locks” on page 4 for more information regarding the use of the SQLSetConnectAttr API to perform XA transactions with the SQL connection as the XA thread of control. If C is specified, any “xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 requests for the specified <i>rmid</i> in this thread will return error XAERR_RMERR.</p> <p>When T is specified for the THDCTL keyword, connection behavior is different for remote connections than it is for local connections. If a remote database is specified for the RDBNAME keyword, a connection to the remote database is opened immediately. This connection is used for all XA requests made in that thread, and all subsequent SQL work requested in that thread, even if the CLI is used to open multiple connection handles. In other words, no more than one physical connection is created for the thread even if the CLI is used to request multiple connections from that thread. If a local database is specified for the RDBNAME keyword, no connection is immediately established, and each connection request establishes a unique physical connection to the database.</p> <p>If this keyword is not specified, THDCTL defaults to T. </p>
TMNAME	<p>A 1- to 10-character name identifying the XA transaction manager. Information is only significant for transaction managers that might require special processing and have worked with the XA resource manager to implement support. This value is displayed on the Display Commitment Definition Status panel when the commitment definition has been opened to act as an XA resource manager. Non-IBM applications must not use a name that starts with the letter Q. The name must adhere to iSeries naming conventions.</p> <p>If this keyword is not specified, TMNAME defaults to blanks.</p>
USER	<p>A 1- to 10-character user profile to be used when accessing the relational database.</p> <p>This value will only be used if a user identifier and password is not specified on the Structured Query Language connection operation that follows the xa_open() request. If USER is not specified and no user profile is specified on the connection operation, the user profile for the connection defaults to the current user profile for the job that makes the connection.</p> <p>If this keyword is not specified, USER defaults to blanks.</p>

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
```

```

char xa_info[1024]=
    "tmname=mytranmgr rdbname=myrdb";

int rmid;
long flags;
int retcode;
extern struct xa_switch_t xa_switch;

retcode =
    xa_switch.xa_open_entry(xa_info, rmid, flags);
}

```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_prepare()— Prepare to Commit an XA Transaction Branch (Transaction Scoped Locks)

Syntax

```

#include <xa.h>

int xa_switch.xa_prepare_entry(XID *xid,
    int rmid, long flags);

```

Default Public Authority: *USE
 Service Program: QTNXADTP
 Threadsafe: Yes

A transaction manager calls **xa_prepare()** to request that a resource manager prepare for commitment any work performed on behalf of **xid*. The resource manager places all resources used in the transaction branch in a state that the changes can be made permanently when it later receives the **xa_commit()** request. All associations for **xid* must have been ended by calling **xa_end()** prior to the prepare request.

Parameters

- *xid** (Input) A pointer to transaction branch identifier. This identifier was generated by the transaction manager when the transaction branch was started.
- rmid** (Input) An integer value that the transaction manager generated when calling **xa_open()**. The rmid identifies the resource manager.
- flags** (Input) The following are valid settings of *flags*.
TMNOFLAGS: 0x00000000L Perform the prepare operation normally.

Authorities

None

Return Value

The following return codes indicate that the resource manager has rolled back the work done on this transaction branch.

100 [XA_RBROLLBACK]

The transaction branch was rolled back for an unspecified reason.

- 101 [XA_RBCOMMFAIL]
A communications failure occurred within the resource manager.
- 102 [XA_RBDEADLOCK]
A deadlock condition was detected within the resource manager.
- 103 [XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.
- 104 [XA_RBOTHER]
The resource manager rolled back the transaction branch for a reason not on this list.
- 105 [XA_RBPROTO]
A protocol error occurred in the resource manager.
- 106 [XA_RBTIMEOUT]
A time-out occurred in the resource manager.
- 107 [XA_RBTRANSIENT]
A transient error was detected in the resource manager.

All other return codes:

- 7 [XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.
- 6 [XAER_PROTO]
xa_prepare() was not successful. Function was called in an improper context.
- 5 [XAER_INVALID]
xa_prepare() was not successful. Incorrect arguments were specified.
- 4 [XAER_NOTA]
The specified *xid* is not known by the resource manager.
- 3 [XAER_RMERR]
xa_prepare() was not successful. The resource manager detected an error when preparing the transaction branch.
- 2 [XAER_ASYNC]
xa_prepare() was not successful. The resource manager does not support asynchronous operations.
- 0 [XA_OK]
xa_prepare() was successful.
- 3 [XA_RDONLY]
The transaction branch was read-only and has been committed.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t xa_switch;

    retcode =
        xa_switch.xa_prepare_entry(xid, rmid, flags);
}
```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_recover()— Recover XA Transaction Branches (Transaction Scoped Locks)

Syntax

```
#include <xa.h>

int xa_switch.xa_recover_entry(XID *xids,
    long count, int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **xa_recover()** during recovery to obtain a list of transaction branches that are currently in a prepared or heuristically completed state. Multiple calls to this function can be made in a single recovery scan. The *flags* parameter defines when a recovery scan should start or end.

Parameters

***xids** (Input) A pointer to an array into which the resource manager places XIDs for transaction branches in prepared or heuristically completed states.

count (Input) The number of *xids* that fit into the *xids* array.

rmid (Input) An integer value that the transaction manager generated when calling **xa_open()**. The *rmid* identifies the resource manager.

flags (Input) The following are valid settings of *flags*. *TMSTARTRSCAN*: 0x01000000L Start a recovery scan and position the cursor to the start of the list. XIDs are returned from that point.

TMENDRSCAN: 0x00800000L End a recovery scan after returning the XIDs. If this flag is used with the *TMSTARTRSCAN* flag, then a single **xa_recover()** call starts and ends the recovery scan.

TMNOFLAGS: 0x00000000L Continue a recovery scan. XIDs are returned starting at the current cursor position. [»](#)

TMREGISTER: 0x00000001L Return XIDs for transaction branches in an idle state, rather than those in prepared or heuristically completed states. This is a proprietary extension of the XA architecture to allow a transaction manager that has failed, and is now restarting, to recover transaction branches that were in an idle state when it failed. If such transaction branches are not recovered, they will persist, possibly holding locks, until an IPL of the system. If this flag is specified when running against a heterogeneous database, the `xa_recover` request will be rejected with return value `XAER_INVAL`. [«](#)

Authorities

None

Return Value

-6	[XAER_PROTO]	
-5	[XAER_INVAL]	<code>xa_recover()</code> was not successful. Function was called in an improper context.
-3	[XAER_RMERR]	<code>xa_recover()</code> was not successful. Incorrect arguments were specified.
≥ 0		<code>xa_recover()</code> was not successful. The resource manager detected an error determining the XIDs to return. The total number of XIDs returned in the <code>xids</code> array.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID  xids[10];
    int  rmid;
    long count=10;
    long flags=TMSTARTRSCAN+TMENDRSCAN;
    int  retcode;
    extern struct xa_switch_t xa_switch;
```

```

retcode =
    xa_switch.xa_recover_entry(xids, count,
                               rmid, flags);
}

```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_rollback()— Roll Back an XA Transaction Branch (Transaction Scoped Locks)

Syntax

```

#include <xa.h>

int xa_switch.xa_rollback_entry(XID *xid,
                                int rmid, long flags);

```

Default Public Authority: *USE
Service Program: QTNXADTP
Threadsafe: Yes

A transaction manager calls **xa_rollback()** to roll back work performed on behalf of the transaction branch. A transaction branch is capable of being rolled back until it has been successfully committed.

Parameters

- *xid** (Input) A pointer to the transaction branch identifier. This identifier was generated by the transaction manager when the transaction branch was started.
- rmid** (Input) An integer value that the transaction manager generated when calling **xa_open()**. The rmid identifies the resource manager.
- flags** (Input) The following are valid settings of *flags*.
TMNOFLAGS: 0x00000000L Perform the rollback operation normally.

Authorities

None

Return Value

The following return codes indicate that the resource manager rolled back the work done on this transaction branch. These values are typically returned when the transaction branch was previously marked rollback-only.

- 100 [XA_RBROLLBACK]
The transaction branch was rolled back for an unspecified reason.
- 101 [XA_RBCOMMFAIL]
A communications failure occurred within the resource manager.
- 102 [XA_RBDEADLOCK]
A deadlock condition was detected within the resource manager.
- 103 [XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.

- 104 [XA_RBOTHER]
The resource manager rolled back the transaction branch for a reason not on this list.
- 105 [XA_RBPROTO]
A protocol error occurred in the resource manager.
- 106 [XA_RBTIMEOUT]
A timeout occurred in the resource manager.
- 107 [XA_RBTRANSIENT]
A transient error was detected in the resource manager.

The following return codes may be returned for any *flags* setting.

- 7 [XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.
- 6 [XAER_PROTO]
xa_rollback() was not successful. Function was called in an improper context.
- 5 [XAER_INVALID]
xa_rollback() was not successful. Incorrect arguments were specified.
- 4 [XAER_NOTA]
The specified *xid* is not known by the resource manager.
- 3 [XAER_RMERR]
xa_rollback() was not successful. The resource manager detected an error when rolling back the transaction.
- 2 [XAER_ASYNC]
xa_rollback() was not successful. The resource manager does not support asynchronous operations.
- 0 [XA_OK]
xa_rollback() was successful.
- 5 [XA_HEURMIX]
Work on the transaction branch was partially committed and partially rolled back.
- 6 [XA_HEURRB]
Work on the transaction branch was heuristically rolled back.
- 7 [XA_HEURCOM]
Work on the transaction branch was heuristically committed.
- 8 [XA_HEURHAZ]
Work on the transaction branch may have been heuristically completed.

Error Messages

The following messages may be sent from this function.

- CPE3418 E Possible APAR condition or hardware failure.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t xa_switch;

    retcode =
        xa_switch.xa_rollback_entry(xid, rmid, flags);
}
```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_start()— Start an XA Transaction Branch (Transaction Scoped Locks)

Syntax

```
#include <xa.h>

int xa_switch.xa_start_entry(XID *xid,
    int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **xa_start()** to inform a resource manager that an application may do work on behalf of a transaction branch. The calling thread becomes associated with the transaction branch.

Parameters

***xid** (Input) A pointer to the transaction branch identifier for the transaction branch that is to be associated with this thread.

rmid (Input) An integer value that the transaction manager generated when calling **xa_open()**. The rmid identifies the resource manager.

flags (Input) Following are the valid settings of flags.

TMJOIN: 0x00200000L Caller is joining an existing transaction branch.

TMRESUME: 0x08000000L Caller is resuming association with a suspended transaction branch.

TMNOWAIT: 0x10000000L Do not associate the transaction branch with the thread if a blocking condition exists.

TMNOFLAGS: 0x00000000L To be used when no other flags are set.

Authorities

None

Return Value

The following return codes may be returned for any *flags* setting.

- 8 [XAER_DUPID]
Neither TMRESUME nor TMJOIN were specified, and the *xid* already exists within the resource manager.
- 7 [XAER_RMFAIL]
An error occurred that makes the resource manager unavailable.
- 6 [XAER_PROTO]
xa_start() was not successful. Function was called in an improper context.
- 5 [XAER_INVALID]
xa_start() was not successful. Incorrect arguments were specified.
- 4 [XAER_NOTA]
TMRESUME or TMJOIN was specified, and the *xid* is not known by the resource manager.
- 3 [XAER_RMERR]
xa_start() was not successful. The resource manager detected an error when associating the transaction branch with the thread.
- 2 [XAER_ASYNC]
xa_start() was not successful. The resource manager does not support asynchronous operations.
- 0 [XA_OK]
xa_start() was successful.
- 4 [XA_RETRY]
TMNOWAIT was set in flags and a blocking condition exists. The thread was not associated with the transaction branch.

The following return codes indicate that TMJOIN or TMRESUME was specified, and the specified transaction branch was not associated with the thread and is marked rollback-only.

- 100 [XA_RBROLLBACK]
The transaction branch was marked rollback-only for an unspecified reason.
- 101 [XA_RBCOMMFAIL]
A communications failure occurred within the resource manager.
- 102 [XA_RBDEADLOCK]
A deadlock condition was detected within the resource manager.
- 103 [XA_RBINTEGRITY]
The resource manager detected a violation of the integrity of its resources.
- 104 [XA_RBOTHER]
The transaction branch was marked rollback-only for a reason not on this list.
- 105 [XA_RBPROTO]
A protocol error occurred in the resource manager.
- 106 [XA_RBTIMEOUT]
A timeout occurred in the resource manager.

A transient error was detected in the resource manager.

Error Messages

The following messages may be sent from this function.

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN:1-872630-24-3, C193 or XO/CAE/91/300), The Open Group.
- X/Open CAE Specification, April 1995, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN:1-85912-094-6, C504), The Open Group.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t xa_switch;

    retcode =
        xa_switch.xa_start_entry(xid, rmid, flags);
}
```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

xa_start_2()—Start an XA Transaction Branch, Extended Version (Transaction Scoped Locks)

Syntax

```
#include <xa.h>

int xa_switch.xa_start_2_entry(XID *xid,
    int rmid, XACTL *ctl, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **xa_start_20** to inform a resource manager that an application may do work on behalf of a transaction branch. The calling thread becomes associated with the transaction branch.

For additional information about parameters, authorities required, return values, and error conditions, see the “`xa_start()`—Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 API.

The `xa_start_2()` function is the same as the `xa_start()` function, with one additional parameter, *ctl*.

***ctl** (Input) A pointer to the following structure.

```
struct xactl_t {
    long flags;                /* valid element flags */
    TRANSACTION_TIMEOUT timeout; /* timeout value */
};
```

Following are the valid settings of *ctl->flags*.

`XAOPTS_TIMEOUT`: 0x00000001L Timeout value is present.

`XAOPTS_NOFLAGS`: 0x00000000L To be used when no optional values are set.

ctl->timeout is the number of seconds before which the resource manager can timeout and rollback the transaction. Type `TRANSACTION_TIMEOUT` is declared in header file *xa.h*.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>
```

```
main() {
    XID *xid;
    int rmid;
    XACTL ctl;
    long flags;
    int retcode;
    extern struct xa_switch_t xa_switch;

    retcode =
        xa_switch.xa_start_2_entry(xid, rmid, &ctl, flags);
}
```

API introduced: V5R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

XA APIs for Job Scoped Locks

The XA APIs for Job Scoped Locks are:

- “`db2xa_close()`—Close an XA Resource Manager (Job Scoped Locks)” on page 34 (Close an XA resource manager (Job Scoped Locks)) is called to close a currently open resource manager in the thread of control.
- “`db2xa_commit()`—Commit an XA Transaction Branch (Job Scoped Locks)” on page 35 (Commit an XA transaction branch (Job Scoped Locks)) is called to commit the work associated with **xid*.
- “`db2xa_complete()`—Test Completion of Asynchronous XA Request (Job Scoped Locks)” on page 36 (Test completion of an asynchronous XA request (Job Scoped Locks)) is called to wait for the completion of an asynchronous operation.
- “`db2xa_end()`—End Work on an XA Transaction Branch (Job Scoped Locks)” on page 36 (End work on an XA transaction branch (Job Scoped Locks)) is called when an application thread of control finishes or needs to suspend work on a transaction branch.

- “db2xa_forget()—Forget an XA Transaction Branch (Job Scoped Locks)” on page 37 (Forget an XA transaction branch (Job Scoped Locks)) is called to forget about a heuristically completed transaction branch.
- “db2xa_open()—Open an XA Resource Manager (Job Scoped Locks)” on page 38 (Open an XA resource manager (Job Scoped Locks)) is called to open the XA resource manager and to prepare it for use in the XA distributed transaction environment.
- “db2xa_prepare()—Prepare to Commit an XA Transaction Branch (Job Scoped Locks)” on page 41 (Prepare to commit an XA transaction branch (Job Scoped Locks)) is called to request that a resource manager prepare for commitment any work performed on behalf of *xid*.
- “db2xa_recover()—Recover XA Transaction Branches (Job Scoped Locks)” on page 42 (Recover XA transaction branches (Job Scoped Locks)) is called during recovery to obtain a list of transaction branches that are currently in a prepared or heuristically completed state.
- “db2xa_rollback()—Roll Back an XA Transaction Branch (Job Scoped Locks)” on page 43 (Roll back an XA transaction branch (Job Scoped Locks)) is called to roll back work performed on behalf of the transaction branch.
- “db2xa_start()—Start an XA Transaction Branch (Job Scoped Locks)” on page 43 (Start an XA transaction branch (Job Scoped Locks)) is called to inform a resource manager that an application may do work on behalf of a transaction branch.

The following exit functions must be provided by a transaction manager for use by the XA resource manager when the XA APIs for Job Scoped Locks are used:

- ax_reg() (Exit program to dynamically register an XA resource manager)
- ax_unreg() (Exit program to dynamically unregister an XA resource manager)

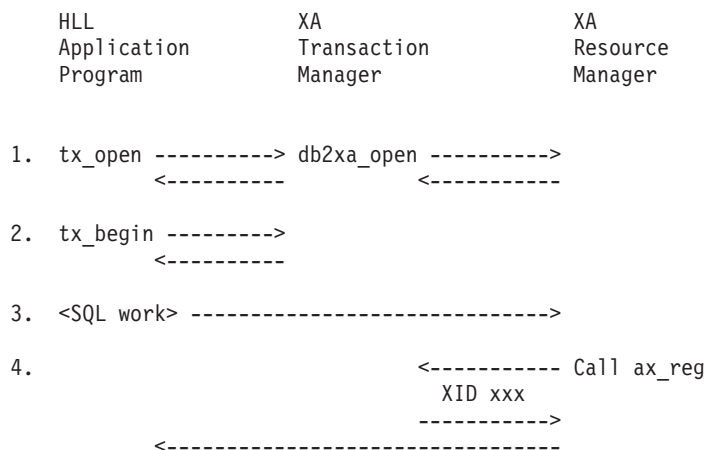
» When using the XA APIs for Job Scoped Locks, the XA thread of control is always considered to be the i5/OS thread from which transactional work is requested, regardless of what SQL connection is used to perform that work. «

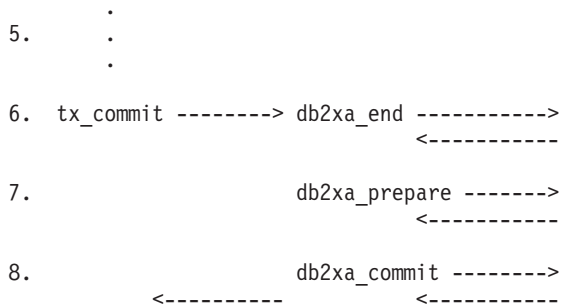
The following example shows the interactions between the application program, transaction manager, and the XA resource manager during a typical transaction when the XA APIs for Job Scoped Locks are used. The actual interactions that occur during a transaction will vary depending on factors such as the following:

- Whether the transaction is committed or rolled back
- Whether the one- or two-phase commit protocol is used with the XA resource manager
- Whether multiple threads are used to perform the work of a transaction branch

Refer to the X/Open XA Specification for details.

Example Using XA APIs for Job Scoped Locks





Notes

1. The application uses the X/Open Transaction Demarcation (TX) **tx_open()** interface to open all the resource managers that are linked with the transaction manager. The transaction manager uses the **db2xa_open()** interface to open an instance of the XA resource manager. The transaction manager may open multiple XA resource managers that will participate in XA transactions. The transaction manager assigns a resource manager identifier (ID) to each resource manager instance. The resource manager ID uniquely identifies the instance within the thread of control in which the application is running. An instance of the XA resource manager can be thought of as an SQL connection to the relational database specified on the **xainfo* parameter of the **db2xa_open()** API.
2. The application uses the TX **tx_begin()** interface to begin a transaction.
3. The application uses SQL interfaces to access resources managed by DB2^(R) UDB for iSeries^(TM).
4. The XA resource manager uses the XA **ax_reg()** interface to dynamically register itself with the transaction manager. The transaction manager returns a transaction branch identifier (XID) that uniquely identifies the transaction branch.
5. The application continues its transaction. It may access other resource managers as appropriate.
6. When the transaction has been completed, the application uses the TX **tx_commit()** interface to commit the work. The transaction manager uses the XA **db2xa_end()** interface to end the transaction branch.
7. The transaction manager uses the XA **db2xa_prepare()** interface to prepare the resources for commitment.
8. The transaction manager uses the XA **db2xa_commit()** interface to commit the resources after all the resource managers involved in the transaction have successfully prepared their resources for commitment. When the commit operation is complete, the application can begin another transaction using the TX **tx_begin()** interface.

Restrictions for XA APIs for Job Scoped Locks

When using the XA APIs for Job Scoped Locks, an application that uses the CLI SQL interfaces must use a single connection to perform all work for a transaction branch. This means that if the XA join function is used so that multiple threads work on a single transaction branch, all the joining threads must use the same CLI connection for that work. Since CLI connection handles cannot be shared across jobs, this means that the XA join function can be used only by threads within a single job when using the CLI. This restriction does not apply when the application uses embedded SQL, or when the XA APIs for Transaction Scoped Locks are used. ➤ The application must not switch connections while a thread is associated with a transaction branch. If the connection is disconnected while a thread is associated with a transaction branch, the transaction branch will implicitly roll back. ⚡

When used with the XA APIs for Job Scoped Locks, some aspects of SQL Server Mode behavior are affected. Traditional SQL Server Mode usage within an application makes a one to one correlation between a connection to the database in the application and to a QSQRVR prestart job in the QSYSWRK subsystem. All SQL requests made in the application using that connection are executed in the correlated QSQRVR job. When the connection is closed, the job is recycled and returned to the prestart job pool.

With XA, an application has the ability to start and use separate transaction branches over a single database connection. When the XA APIs for Job Scoped Locks are used to start a new transaction branch using a connection that was earlier used for a different transaction branch that has not yet been completed (committed or rolled back), the new transaction branch is assigned its own QSQRVR job. This means a single connection can be related to multiple QSQRVR jobs. When a transaction branch that requires a new QSQRVR job completes, that QSQRVR job is dissociated from the connection, recycled and returned to the prestart job pool.

If embedded SQL is used and the native DB2 UDB for iSeries security mechanisms are used, the transaction manager must ensure that all work on a transaction branch is performed by jobs or threads using the same user profile. In other words, if the XA join function is used, every joining thread or job must use the same user profile as the thread or job that started the transaction branch; otherwise, a security exposure will exist. This security consideration does not exist when using the XA APIs for Transaction Scoped Locks because the one to one correlation between the connection and the QSQRVR job is always maintained, regardless of what transaction branch is being worked on.

While this model works well for isolating transactions, the environment may provide some extra work on behalf of the application. Since separate and distinct jobs are in use for each transaction branch, any job/process-scoped resources setup while under one transaction branch will be unavailable once the application has switched to a different transaction branch. A list of the known limitations and restrictions when using this support is included below. This list is not guaranteed to be comprehensive.

The following example demonstrates a scenario where these restrictions may be encountered.

1. db2xa_open()
2. SQL Connect. This may be skipped if connection is to start implicitly when the first embedded SQL request is made.
3. Set up to have ax_reg() return TM_OK for XID1 when SQL work is requested.
4. SQL statements to perform work. The first statement causes transaction branch XID1 to be created. The work for XID1 is done within SQL Server Mode Job: xxxxxx/QUSER/QSQRVR).
5. db2xa_end() with flag TMSUSPEND for XID1.
6. Set up to have ax_reg() return TM_OK for XID2 when SQL work is requested.
7. SQL statements to perform work. The first statement causes transaction branch XID2 to be created. The work for XID2 is done within SQL Server Mode Job: yyyyyy/QUSER/QSQRVR).
8. db2xa_end() with flag TMSUCCESS for XID2.
9. Set up to have ax_reg() return TM_RESUME for XID1 when SQL work is requested.
10. SQL statements to perform work . The first statement causes transaction branch XID1 to be resumed. The work for XID1 is done within SQL Server Mode Job: xxxxxx/QUSER/QSQRVR).
11. db2xa_end() with flag TMSUCCESS for XID1.
12. db2xa_prepare() XID1. This may be requested from any thread.
13. db2xa_commit() XID1. This may be requested from any thread.
14. db2xa_prepare() XID2. This may be requested from any thread.
15. db2xa_commit() XID2. This may be requested from any thread.

SQL prepared statements

When an application prepares an SQL statement, the resulting statement is stored in a job-scoped system space. This means that, for the example above, statements prepared while working on transaction branch XID1 are not available while working on transaction branch XID2, because the SQL work for the two transaction branches is done in separate QSQRVR jobs. If the application attempts to use a prepared statement that is not available, the failure symptom would be SQLCODE = -518. (SQL0518 - Prepared statement &1 not found.)

SQL Cursors

SQL cursors are **»** thread-scoped resources, so they also **«** are not available to the application after switching to a new transaction branch. If an application opens an SQL cursor and changes transaction branches, the cursor may remain open in the QSQSRVR job related to the previous transaction branch depending on how that branch was ended (see “SQLHOLD Values” on page 39). However, the cursor will not be available while working on the new transaction branch. If and when the original transaction branch is resumed, open cursors related to that transaction branch would again become available. Attempting to reference a cursor while executing under a transaction branch other than the one under which the cursor was opened, will result in a failure of SQLCODE = -501. (SQL0501 - Cursor &1 not open.)

Result Sets

When calling a stored procedure that returns result set(s), the application needs to take care to fully process the result sets before changing to a different transaction branch. SQL CLI services that return information about the status of a result set, could return incorrect information if not used in this manner. Examples of SQL CLI APIs that return information based on interim results are SQLNumResultCols(), SQLDescribeCol(), SQLColAttributes() and SQLDescribeParam().

SQL CLI APIs like SQLFetch() and SQLFetchScroll(), which deal directly with the SQL result set cursor, would fail with SQLCODE = -502. (SQL0502 - Cursor &1 already open.)

SET PATH statement

The SET PATH SQL statement allows the application to designate a path to use for unqualified library access to SQL stored procedures, SQL triggers and SQL UDFs within a dynamic statement. The path is a job-scoped resource, and therefore not available after changing transaction branches. The application should repeat any SET PATH statements after a transaction branch change, if the path will still be needed.

Other SQL considerations

Applications should not change transaction branches while running within an SQL Stored Procedure, an SQL User Defined Function (UDF) or an SQL Trigger program. Results would be unpredictable and no anticipated failure information is available.

» Embedded SQL applications that use the QSQCHGDC() system API or SET SCHEMA SQL statement to set up the Dynamic Default Connection will not function correctly because they will not affect the SQL Server Mode job. **«** This has always been a restriction of the SQL Server Mode environment. If encountered, the failure symptom seen by the application would be SQLCODE = -204. (&1 in &2 type *&3 not found.)

Note that SQL CLI users that set the default library using the SQLSetConnectAttr() API with the SQL_ATTR_DBC_DEFAULT_LIB connection attribute will continue to work. SQL CLI connection attributes are still in place after moving to a different transaction branch.

Top | UNIX-Type APIs | APIs by category

db2xa_close()—Close an XA Resource Manager (Job Scoped Locks)

Syntax

```
#include <xa.h>

int db2xa_switch.xa_close_entry(char *xa_info,
    int rmid, long flags);
```


Default Public Authority: *USE
Service Program: QTNXADTP
Threadsafe: Yes

A transaction manager calls **db2xa_close()** to close a currently open resource manager in the thread of control. After this call, the resource manager cannot participate in global transactions on behalf of the calling thread until it is reopened.

For additional information about parameters, authorities required, return values, and error conditions, see the “[xa_close\(\)— Close an XA Resource Manager \(Transaction Scoped ,Locks\)](#)” on page 7 API.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    char *xa_info;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t db2xa_switch;

    retcode =
        db2xa_switch.xa_close_entry(xa_info, rmid, flags);
}
```

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

db2xa_commit()—Commit an XA Transaction Branch (Job Scoped Locks)

Syntax

```
#include <xa.h>

int db2xa_switch.xa_commit_entry(XID *xid,
    int rmid, long flags);
```

Default Public Authority: *USE
Service Program: QTNXADTP
Threadsafe: Yes

A transaction manager calls **db2xa_commit()** to commit the work associated with **xid*. All changes that were made to resources managed by DB2 UDB for iSeries during the transaction branch are made permanent.

For additional information about parameters, authorities required, return values, and error conditions, see the “[xa_commit\(\)— Commit an XA Transaction Branch \(Transaction Scoped Locks\)](#)” on page 9 API.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
```

```

XID *xid;
int rmid;
long flags;
int retcode;
extern struct xa_switch_t db2xa_switch;

retcode =
    db2xa_switch.xa_commit_entry(xid, rmid, flags);
}

```

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

db2xa_complete()—Test Completion of Asynchronous XA Request (Job Scoped Locks)

Syntax

```

#include <xa.h>

int db2xa_switch.xa_complete_entry(int *handle,
    int *retval, int rmid, long flags)

```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **db2xa_complete()** to wait for the completion of an asynchronous operation. Asynchronous operations are not supported by the DB2 UDB for iSeries resource manager. This function is provided only for compliance with the X/Open XA Specification.

For additional information about parameters, authorities required, return values, and error conditions, see the “[xa_complete\(\)—Test Completion of Asynchronous XA Request \(Transaction Scoped Locks\)](#)” on page 11 API.

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

db2xa_end()—End Work on an XA Transaction Branch (Job Scoped Locks)

Syntax

```

#include <xa.h>

int db2xa_switch.xa_end_entry(XID *xid, int rmid,
    long flags);

```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **db2xa_end()** when an application thread of control finishes or needs to suspend work on a transaction branch. When **db2xa_end()** successfully returns, the calling thread of control is no longer associated with the transaction branch, but the branch still exists.

SQL cursors used while the thread was associated with this transaction branch may be closed. Refer to the SQLHOLD keyword description in the usage notes of the “db2xa_open()—Open an XA Resource Manager (Job Scoped Locks)” on page 38 API for details.

For additional information about parameters, authorities required, return values, and error conditions, see the “xa_end()—End Work on an XA Transaction Branch (Transaction Scoped Locks)” on page 12 API.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t db2xa_switch;

    retcode =
        db2xa_switch.xa_end_entry(xid, rmid, flags);
}
```

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

db2xa_forget()—Forget an XA Transaction Branch (Job Scoped Locks)

Syntax

```
#include <xa.h>

int db2xa_switch.xa_forget_entry(XID *xid,
    int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **db2xa_forget()** to forget about a heuristically completed transaction branch. After this call, the **xid* is no longer valid.

For additional information about parameters, authorities required, and error conditions, see the “xa_forget()—Forget an XA Transaction Branch (Transaction Scoped Locks)” on page 15 API.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t db2xa_switch;

    retcode =
        db2xa_switch.xa_forget_entry(xid, rmid, flags);
}
```

db2xa_open()—Open an XA Resource Manager (Job Scoped Locks)

Syntax

```
#include <xa.h>

int db2xa_switch.xa_open_entry(char *xa_info,
    int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **db2xa_open()** to open the XA resource manager and to prepare it for use in the XA distributed transaction environment. This function must be called before any other resource manager (*db2xa_*) calls are made.

For additional information about parameters, authorities required, return values, and error conditions, see the “*xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)*” on page 16 API.



Authorities

In addition to those documented for the “*xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)*” on page 16 API, the following authorities are required.

Exit Program Authority (specified via the xa_info parameter SRVPGM keyword)

*USE

Exit Program Library Authority (specified via the xa_info parameter SRVPGM keyword)

*EXECUTE

Journal Authority (if specified via the xa_info parameter DFTJRN keyword)

*OBJOPR *ADD

Journal Library Authority (if specified via the xa_info parameter DFTJRN keyword)

*EXECUTE



Usage Notes

The usage notes for the “*xa_open()—Open an XA Resource Manager (Transaction Scoped Locks)*” on page 16 API apply to this API with the following differences.

- Additional *xa_info* keywords shown in xainfo String Keywords and Values (page 38) are allowed.
- The LOCKWAIT *xa_info* keyword is not allowed.

xainfo String Keywords and Values

Keyword Name	Keyword Value
DFTJRN	<p>Default Journal. See the online help for the DFTJRN keyword of the STRCMTCTL CL command for a description of the effect of this keyword. The journal should be specified as the journal's library, concatenated with a '/', concatenated with the journal's name (for example, MYLIB/MYJRN). Both the library and journal name must follow iSeries conventions for naming system objects.</p> <p>The special value *NONE is supported for default journal.</p> <p>The special value *LIBL is accepted for the library portion of the default journal and is the default if the library portion is not specified.</p> <p>If this keyword is not specified, no default journal is used.</p> <p>If this keyword is specified but unresolvable, [XAER_INVAL] is returned.</p>
OMTJRNE	<p>Omit Journal Entries. See the online help for the OMTJRNE keyword of the STRCMTCTL CL command for a description of the effect of this keyword.</p> <p><i>N</i> Corresponds to the STRCMTCTL OMTJRNE value *NONE.</p> <p><i>L</i> Corresponds to the STRCMTCTL OMTJRNE value *LUWID.</p> <p>If this keyword is not specified, OMTJRNE defaults to <i>N</i>.</p>
SQLHOLD	<p>SQL HOLD value. Whether SQL cursors are closed during some XA operations. Refer to "SQLHOLD Values" for detailed information about this keyword.</p> <p>If this keyword is not specified, SQLHOLD defaults to <i>A</i>.</p>
SRVPGM	<p>The name of a library qualified service program that contains functions <code>ax_reg()</code> and <code>ax_unreg()</code> to be called by the resource manager to register and unregister itself with the transaction manager. The service program should be specified as the program's library, concatenated with a '/', concatenated with the program's name (for example, TMLIB/TMPGM). Both the library and program name must follow iSeries conventions for naming system objects.</p> <p>The special value *LIBL is supported for the library portion of the service program and is the default if the library portion is not specified.</p> <p>This is a required keyword. If this keyword is not specified, or is unresolvable, [XAER_INVAL] is returned.</p> <p>See <code>ax_reg()</code>—Exit Program to Dynamically Register an XA Resource Manager and <code>ax_unreg()</code>—Exit Program to Dynamically Unregister an XA Resource Manager for details on these service functions.</p>

SQLHOLD Values

This section documents how the SQLHOLD keyword value affects SQL cursors during the following XA operations (other XA operations do not affect cursors):

- `db2xa_end()` *unless the TMSUSPEND flag is specified*
- `db2xa_commit()`
- `db2xa_rollback()`

This applies only to cursors associated with the connection that is used for the transaction branch affected by the XA operation. As shown below, cursors declared WITH HOLD are treated differently in some cases than those not declared WITH HOLD. Note that cursors can be declared WITH HOLD only when embedded SQL is used. CLI cursors are not declared WITH HOLD.

- A* Cursors are affected by XA operations as follows:
- db2xa_end() with the TMSUCCESS or TMFAIL flag:
 - All cursors are closed.
 - db2xa_commit():
 - Cursors are not affected since db2xa_end() already closed them.
 - db2xa_rollback():
 - Cursors are not affected since db2xa_end() already closed them.
- E* Cursors are affected by XA operations as follows:
- db2xa_end() with the TMSUCCESS or TMFAIL flag:
 - Cursors declared WITH HOLD are held open.
 - Cursors not declared WITH HOLD are closed.
 - db2xa_commit():
 - Cursors declared WITH HOLD are held open.
 - Cursors not declared WITH HOLD are closed.
 - db2xa_rollback():
 - All cursors are closed.
- L* Cursors are affected by XA operations as follows:
- db2xa_end() with the TMSUCCESS or TMFAIL flag:
 - All cursors are held open.
 - db2xa_commit():
 - If the relational database resides on an iSeries system:
 - All cursors are left open.
 - If the relational database does not reside on an iSeries system:
 - Cursors declared WITH HOLD are left open.
 - Cursors not declared WITH HOLD are closed.
 - db2xa_rollback():
 - If the relational database resides on an iSeries system:
 - All cursors are left open.
 - If the relational database does not reside on an iSeries system:
 - All cursors are closed.
- N* Cursors are affected by XA operations as follows:
- db2xa_end() with the TMSUCCESS or TMFAIL flag:
 - All cursors are held open.
 - db2xa_commit():
 - Cursors declared WITH HOLD are held open.
 - Cursors not declared WITH HOLD are closed.
 - db2xa_rollback():
 - All cursors are closed.

- Y Cursors are affected by XA operations as follows:
- `db2xa_end()` with the TMSUCCESS or TMFAIL flag:
 - All cursors are held open.
 - `db2xa_commit()`:
 - If the relational database resides on an iSeries system:
 - All cursors are left open.
 - If the relational database does not reside on an iSeries system:
 - The `db2xa_commit()` operation will fail. This value should not be used with relational databases that do not reside on an iSeries system.
 - `db2xa_rollback()`:
 - If the relational database resides on an iSeries system:
 - All cursors are left open.
 - If the relational database does not reside on an iSeries system:
 - The `db2xa_rollback()` operation will fail. This value should not be used with relational databases that do not reside on an iSeries system.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {

    char xa_info[1024]=
        "tmname=mytranmgr srvgm=tmlib/tmserv rdbname=myrdb";

    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t db2xa_switch;

    retcode =
        db2xa_switch.xa_open_entry(xa_info, rmid, flags);
}
```

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

db2xa_prepare()—Prepare to Commit an XA Transaction Branch (Job Scoped Locks)

Syntax

```
#include <xa.h>

int db2xa_switch.xa_prepare_entry(XID *xid,
    int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls `db2xa_prepare()` to request that a resource manager prepare for commitment any work performed on behalf of `*xid`. The resource manager places all resources used in the transaction

branch in a state that the changes can be made permanently when it later receives the **db2xa_commit()** request. All associations for **xid* must have been ended by calling **db2xa_end()** prior to the prepare request.

For additional information about parameters, authorities required, return values, and error conditions, see the “**xa_prepare()**— Prepare to Commit an XA Transaction Branch (Transaction Scoped Locks)” on page 21 API.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t db2xa_switch;

    retcode =
        db2xa_switch.xa_prepare_entry(xid, rmid, flags);
}
```

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

db2xa_recover()—Recover XA Transaction Branches (Job Scoped Locks)

Syntax

```
#include <xa.h>

int db2xa_switch.xa_recover_entry(XID *xids,
    long count, int rmid, long flags);
```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **db2xa_recover()** during recovery to obtain a list of transaction branches that are currently in a prepared or heuristically completed state. Multiple calls to this function can be made in a single recovery scan. The *flags* parameter defines when a recovery scan should start or end.

For additional information about parameters, authorities required, return values, and error conditions, see the “**xa_recover()**— Recover XA Transaction Branches (Transaction Scoped Locks)” on page 23 API.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include <xa.h>

main() {
    XID xids[10];
    int rmid;
    long count=10;
    long flags=TMSTARTRSCAN+TMENDRSCAN;
    int retcode;
```



```

extern struct xa_switch_t db2xa_switch;

retcode =
    db2xa_switch.xa_recover_entry(xids, count,
                                  rmid, flags);
}

```

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

db2xa_rollback()—Roll Back an XA Transaction Branch (Job Scoped Locks)

Syntax

```

#include <xa.h>

int db2xa_switch.xa_rollback_entry(XID *xid,
                                   int rmid, long flags);

```

Default Public Authority: *USE

Service Program: QTNXADTP

Threadsafe: Yes

A transaction manager calls **db2xa_rollback()** to roll back work performed on behalf of the transaction branch. A transaction branch is capable of being rolled back until it has been successfully committed.

For additional information about parameters, authorities required, return values, and error conditions, see the “[xa_rollback\(\)— Roll Back an XA Transaction Branch \(Transaction Scoped Locks\)](#)” on page 25 API.

Example

See Code disclaimer information for information pertaining to code examples.

```

#include <xa.h>

main() {
    XID *xid;
    int rmid;
    long flags;
    int retcode;
    extern struct xa_switch_t db2xa_switch;

    retcode =
        db2xa_switch.xa_rollback_entry(xid, rmid, flags);
}

```

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

db2xa_start()—Start an XA Transaction Branch (Job Scoped Locks)

Syntax

```

#include <xa.h>

int db2xa_switch.xa_start_entry(XID *xid,
                                 int rmid, long flags);

```

Default Public Authority: *USE
Service Program: QTNXADTP
Threadsafe: Yes

A transaction manager calls **db2xa_start()** to inform a resource manager that an application may do work on behalf of a transaction branch. When using the XA APIs for Job Scoped Locks, the XA resource manager does not use this function. It dynamically registers work done on behalf of a transaction by using the **ax_reg()** function. This function is provided only for compliance with the X/Open XA Specification.

For additional information about parameters, authorities required, return values, and error conditions, see the “**xa_start()**— Start an XA Transaction Branch (Transaction Scoped Locks)” on page 27 API.

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) IBM 2006. Portions of this code are derived from IBM Corp. Sample Programs. (C) Copyright IBM Corp. 1998, 2006. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This Application Programming Interfaces (API) publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

Advanced 36
Advanced Function Printing
Advanced Peer-to-Peer Networking
AFP
AIX
AS/400
COBOL/400
CUA
DB2
DB2 Universal Database
Distributed Relational Database Architecture
Domino
DPI
DRDA
eServer
GDDM
IBM
Integrated Language Environment
Intelligent Printer Data Stream
IPDS
i5/OS
iSeries
Lotus Notes
MVS
Netfinity
Net.Data
NetView
Notes
OfficeVision
Operating System/2
Operating System/400
OS/2
OS/400
PartnerWorld
PowerPC
PrintManager
Print Services Facility
RISC System/6000
RPG/400
RS/6000
SAA
SecureWay
System/36
System/370
System/38
System/390
VisualAge
WebSphere
xSeries

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Terms and Conditions

Permissions for the use of these Publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these Publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these Publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations. IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE



Printed in USA