UNIX-Type APIs (V5R2)

XA APIs

Table of Contents

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

Header Files for UNIX-Type Functions
Errno Values for UNIX-Type Functions

XA APIs

DB2 UDB for iSeries provides two sets of XA APIs:

- XA APIs for Transaction Scoped Locks
- XA APIs for Job Scoped Locks

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-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 is called xa_switch. The switch for the XA APIs for Job Scoped Locks 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.

≫If you are running XA transactions against a database that resides on the local system, you should use the XA APIs for Transaction Scoped Locks. These APIs have fewer restrictions than the XA APIs for Job Scoped Locks, 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 are used.

If you are running against a database that resides on a remote system, the XA APIs for Job Scoped Locks must be used.≪

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

Restrictions

≫Transactions that require the use of an XA resource manager must be performed in SQL server jobs. 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()** and **db2xa_open()** APIs will set the server mode attribute to *YES if the attribute has not already been set. ≪For additional information about SQL server job, 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.

X/Open applications are only allowed to use SQL interfaces to access resources managed by DB2 UDB for iSeries. Both the embedded and call level interface (CLI) SQL interfaces are supported. ≫Local relational databases may be used by the application when running with the XA APIs for Transaction Scoped Locks or the XA APIs for Job Scoped Locks. Local databases include those defined for an Indpendent ASP. Remote relational databases may be used by the application only when running with the XA APIs for Job Scoped Locks. ≪When using 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.

The following interfaces are not supported for use by an X/Open application:

- Control language (CL) or high-level language (HLL) interfaces for local files or distributed data management (DDM) files.
- The Process Extended Dynamic SQL (QSQPRCED) API.
- The Query (QQQQRY) API.
- The commitment control API interfaces documented in the Journal and Commit APIs part.

It is expected that most transaction managers will use the same user profile for all SQL connections. ≫If the **xa_open** or **db2xa_open** 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()** or **db2xa_open()** 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.

Top | UNIX-Type APIs | APIs by category

XA APIs for Transaction Scoped Locks

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

- »xa_close() (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)) commits the work associated with **xid*.«
- »xa_complete() (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)) 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)) is called to forget about a heuristically completed transaction branch.«
- »xa_open() (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)) 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)) 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)) is called to roll back work performed on behalf of the transaction branch.«
- »xa_start() (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)) informs a resource manager that an application may do work on behalf of a transaction branch.«

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

```
HLL                  XA                     XA
Application          Transaction            Resource
Program              Manager                Manager
```

```
1.  tx_open ----------> xa_open ------------->
           <----------          <-----------

                                   XID xxx
2.  tx_begin ---------> xa_start ------------>
           <----------          <-----------

3.  <SQL work> ----------------------------->
           <---------------------------------

4.      .
        .
        .

5.  tx_commit --------> xa_end -------------->
                                <-----------

6.                      xa_prepare ---------->
                                <-----------

7.                      xa_commit ----------->
           <----------          <-----------
```

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.

»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_INVAL]

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

```
#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

»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_INVAL]

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

```
#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

»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

»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_INVAL]

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

```
#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

»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_INVAL] **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**
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

```
#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

»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_INVAL]
	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_INVAL] is returned.
    - The xa_info string value is treated as character data and is not converted.
    - The return value [XAER_INVAL] will be returned if a keyword is specified that is not documented in Figure 1.

Figure 1. xainfo String Keywords and Values

Keyword Name	**Keyword Value**
LOCKWAIT	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. If not specified, lock wait time values specified by other system interfaces are used. The maximum value that may be specified is 999999999.
PASSWORD	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 db2_xaopen() 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. If this keyword is not specified, PASSWORD defaults to 10 blanks.

PWDLEN	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. 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.
RDBNAME	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. This is a required keyword. If this keyword is not specified, [XAER_INVAL] is returned. 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.
TMNAME	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. If this keyword is not specified, TMNAME defaults to blanks.
USER	A 1- to 10-character user profile to be used when accessing the relational database. 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. If this keyword is not specified, USER defaults to blanks.

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

```
#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

»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_INVAL]

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

```
#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

»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.

Authorities

None

Return Value

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

```
#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

»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 is 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_INVAL]

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

```
#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

»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_INVAL]

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.

107 [XA_RBTRANSIENT]

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

```
#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

»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_2()** 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() 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

```
#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

XA APIs for Job Scoped Locks

The XA APIs for Job Scoped Locks are:

- db2xa_close() (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)) is called to commit the work associated with **xid*.
- db2xa_complete() (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)) 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)) is called to forget about a heuristically completed transaction branch.
- db2xa_open() (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)) 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)) 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)) is called to roll back work performed on behalf of the transaction branch.
- db2xa_start() (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)

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«

```
HLL                  XA                     XA
Application          Transaction            Resource
Program              Manager                Manager
```

```
1.  tx_open ----------> db2xa_open ---------->
            <----------            <----------

2.  tx_begin --------->
            <----------

3.  <SQL work> ------------------------------>

4.                                 <---------- Call ax_reg
                                     XID xxx
                                   ---------->
            <---------------------------------


        .
5.      .
        .

6.  tx_commit --------> db2xa_end ----------->
                                   <----------

7.                      db2xa_prepare ------->
                                   <----------

8.                      db2xa_commit -------->
            <----------            <----------
```

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 UDB for iSeries.

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. «

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 QSQSRVR prestart job in the QSYSWRK subsystem. All SQL requests made in the application using that connection are executed in the correlated QSQSRVR 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 QSQSRVR job. This means a single connection can be related to multiple QSQSRVR jobs. When a transaction branch that requires a new QSQSRVR job completes, that QSQSRVR 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 QSQSRVR 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/QSQSRVR).

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/QSQSRVR).

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/QSQSRVR).

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 QSQSRVR 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 also job-scoped resources, so they 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). 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 to set up the Dynamic Default Connection will not function correctly because the QSQCHGDC() 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 transacation branch.

---

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() API. «

Example

```
#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

---

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() API. «

Example

```
#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

---

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() API. «

API introduced: V4R3

---

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() API for details.

» For additional information about parameters, authorities required, return values, and error conditions, see the xa_end() API. «

Example

```
#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

---

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() API. «

Example

```
#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);
}
```

API introduced: V4R3

---

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() API. «

Usage Notes

» The usage notes for the xa_open() API apply to this API with the following differences. «

- Additional xa_info keywords shown in Figure 1-4 are allowed.
- » The LOCKWAIT xa_info keyword is not allowed. «

Figure 1-4. xainfo String Keywords and Values

Keyword Name	**Keyword Value**
DFTJRN	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. The special value *NONE is supported for default journal. 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. If this keyword is not specified, no default journal is used. If this keyword is specified but unresolvable, [XAER_INVAL] is returned.

OMTJRNE	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. *N* Corresponds to the STRCMTCTL OMTJRNE value *NONE. *L* Corresponds to the STRCMTCTL OMTJRNE value *LUWID. If this keyword is not specified, OMTJRNE defaults to **N**.
SQLHOLD	SQL HOLD value. Whether SQL cursors are closed during some XA operations. Refer to SQLHOLD Values for detailed information about this keyword. If this keyword is not specified, SQLHOLD defaults to **A**.
SRVPGM	The name of a library qualified service program that contains functions **ax_reg()** and **ax_unreg()** 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. 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. This is a required keyword. If this keyword is not specified, or is unresolvable, [XAER_INVAL] is returned. See ax_reg()--Exit Program to Dynamically Register an XA Resource Manager and ax_unreg()--Exit Program to Dynamically Unregister an XA Resource Manager for details on these service functions.

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

```
#include <xa.h>

main() {

  char xa_info[1024]=
       "tmname=mytranmgr srvpgm=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

---

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() API. «

Example

```
#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

---

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() API. «

Example

```
#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

---

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 is has been successfully committed.

» For additional information about parameters, authorities required, return values, and error conditions, see the xa_rolback() API. «

Example

```
#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

---

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() API. «

API introduced: V4R3

---

ax_reg()--Exit Program to Dynamically Register an XA Resource Manager »(Job Scoped Locks) «

Syntax

```
#include <xa.h>

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

Threadsafe: Conditional; see Usage Notes.

The XA resource manager calls **ax_reg()** to inform a transaction manager that it is about to do work on behalf of an application in a thread of control. The transaction manager needs to tell the resource manager whether or not that work should be performed on behalf of a transaction branch. If the work is part of a transaction branch, the transaction manager will return the transaction branch identifier in *xid. If the work is not part of a transaction branch, the transaction manager will return the **NULLXID** in *xid.

The XA resource manager indicates that it uses dynamic registration by setting the TMREGISTER value in the flags element of its **xa_switch_t** structure.

The name of the service program that contains **ax_reg()** and **ax_unreg()** must be provided to the XA resource manager in the *xa_info parameter of the **db2xa_open()** call.

Parameters

rmid

(Input) The resource manager identifier that was generated by a transaction manager when the resource manager was opened.

***xid**

(Input) A pointer to the buffer where the transaction manager will store the generated transaction branch identifier. This identifier is associated with work done in the calling thread of control or with a **NULLXID**, which indicates that work is being done outside a transaction branch.

flags

(Input) The flags argument must be set to this value. TMNOFLAGS: 0x00000000L No flags are defined for this function.

Authorities

None

Return Value

-3 [TMER_PROTO]

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

-2 [TMER_INVAL]

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

-1 [TMER_TMERR]

ax_reg() was not successful. The transaction manager detected an error when registering the resource.

0 [TM_OK]

ax_reg() was successful.

1 [TM_RESUME]

The resource manager should resume work on a previously suspended transaction branch. If the resource manager does not recognize the *xid, it will return a failure indication to the application.

2 [TM_JOIN]

The resource manager is joining the work of an existing transaction branch. If the resource manager does not recognize the *xid, it will return a failure indication to the application.

Usage Notes

1. This function must be threadsafe if the transaction manager calls the XA APIs in a multithreaded job.
2. Refer to Restrictions in the introduction to the XA APIs for restrictions when using the TM_JOIN return value.

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.

Exit program introduced: V4R3

---

ax_unreg()--Exit Program to Dynamically Unregister an XA Resource Manager »(Job Scoped Locks)«

Syntax

```
#include <xa.h>

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

Threadsafe: Conditional; see Usage Notes.

The XA resource manager calls **ax_unreg()** to inform a transaction manager that it has completed work on a local transaction. The local transaction was started after receiving a NULLXID from **ax_reg()**.

The XA resource manager indicates that it uses the dynamic registration facility by setting the TMREGISTER value in the flags element of its xa_switch_t structure.

The name of the service program that contains **ax_reg()** and **ax_unreg()** must be provided to the XA resource manager in the *xa_info parameter of the **db2xa_open()** call.

Parameters

rmid

(Input) The resource manager identifier that was generated by a transaction manager when the resource manager was opened.

flags

(Input) The flags argument must be set to this value. TMNOFLAGS: 0x00000000L No flags are defined for this function.

Authorities

None

Return Value

-*3* [TMER_PROTO]

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

-*2* [TMER_INVAL]

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

-1 [TMER_TMERR]

ax_unreg() was not successful. The transaction manager detected an error when unregistering the resource.

0 [TM_OK]

ax_unreg() was successful.

Usage Notes

1. This function must be threadsafe if the transaction manager calls the XA APIs in a multithreaded job.

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.

Exit program introduced: V4R3

Top | UNIX-Type APIs | APIs by category

Header Files for UNIX-Type Functions

Programs using the UNIX-type functions must include one or more header files that contain information needed by the functions, such as:

- Macro definitions
- Data type definitions
- Structure definitions
- Function prototypes

The header files are provided in the QSYSINC library, which is optionally installable. Make sure QSYSINC is on your system before compiling programs that use these header files. For information on installing the QSYSINC library, see Data structures and the QSYSINC Library.

The table below shows the file and member name in the QSYSINC library for each header file used by the UNIX-type APIs in this publication.

Name of Header File	Name of File in QSYSINC	Name of Member
arpa/inet.h	ARPA	INET
arpa/nameser.h	ARPA	NAMESER
bse.h	H	BSE
bsedos.h	H	BSEDOS
bseerr.h	H	BSEERR
dirent.h	H	DIRENT
errno.h	H	ERRNO
fcntl.h	H	FCNTL
grp.h	H	GRP
»inttypes.h	H	INTTYPES«
limits.h	H	LIMITS
»mman.h	H	MMAN«
netdbh.h	H	NETDB
»netinet/icmp6.h	NETINET	ICMP6«
net/if.h	NET	IF
netinet/in.h	NETINET	IN
netinet/ip_icmp.h	NETINET	IP_ICMP
netinet/ip.h	NETINET	IP
»netinet/ip6.h	NETINET	IP6«
netinet/tcp.h	NETINET	TCP
netinet/udp.h	NETINET	UDP
netns/idp.h	NETNS	IDP
netns/ipx.h	NETNS	IPX
netns/ns.h	NETNS	NS
netns/sp.h	NETNS	SP
net/route.h	NET	ROUTE
nettel/tel.h	NETTEL	TEL

os2.h	H	OS2
os2def.h	H	OS2DEF
pwd.h	H	PWD
Qlg.h	H	QLG
qp0lflop.h	H	QP0LFLOP
»qp0ljrnl.h	H	QP0LJRNL«
»qp0lror.h	H	QP0LROR«
Qp0lstdi.h	H	QP0LSTDI
qp0wpid.h	H	QP0WPID
qp0zdipc.h	H	QP0ZDIPC
qp0zipc.h	H	QP0ZIPC
qp0zolip.h	H	QP0ZOLIP
qp0zolsm.h	H	QP0ZOLSM
qp0zripc.h	H	QP0ZRIPC
qp0ztrc.h	H	QP0ZTRC
qp0ztrml.h	H	QP0ZTRML
qp0z1170.h	H	QP0Z1170
»qsoasync.h	H	QSOASYNC«
qtnxaapi.h	H	QTNXAAPI
qtnxadtp.h	H	QTNXADTP
qtomeapi.h	H	QTOMEAPI
qtossapi.h	H	QTOSSAPI
resolv.h	H	RESOLVE
semaphore.h	H	SEMAPHORE
signal.h	H	SIGNAL
spawn.h	H	SPAWN
ssl.h	H	SSL
sys/errno.h	H	ERRNO
sys/ioctl.h	SYS	IOCTL
sys/ipc.h	SYS	IPC
sys/layout.h	H	LAYOUT
sys/limits.h	H	LIMITS
sys/msg.h	SYS	MSG
sys/param.h	SYS	PARAM
»sys/resource.h	SYS	RESOURCE«
sys/sem.h	SYS	SEM
sys/setjmp.h	SYS	SETJMP
sys/shm.h	SYS	SHM
sys/signal.h	SYS	SIGNAL
sys/socket.h	SYS	SOCKET
sys/stat.h	SYS	STAT
sys/statvfs.h	SYS	STATVFS

sys/time.h	SYS	TIME
sys/types.h	SYS	TYPES
sys/uio.h	SYS	UIO
sys/un.h	SYS	UN
sys/wait.h	SYS	WAIT
»ulimit.h	H	ULIMIT«
unistd.h	H	UNISTD
utime.h	H	UTIME

You can display a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to display the **unistd.h** header file using the Source Entry Utility editor, enter the following command:

  ```
  STRSEU SRCFILE(QSYSINC/H) SRCMBR(UNISTD) OPTION(5)
  ```

- Using the Display Physical File Member command. For example, to display the **sys/stat.h** header file, enter the following command:

  ```
  DSPPFM FILE(QSYSINC/SYS) MBR(STAT)
  ```

You can print a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to print the **unistd.h** header file using the Source Entry Utility editor, enter the following command:

  ```
  STRSEU SRCFILE(QSYSINC/H) SRCMBR(UNISTD) OPTION(6)
  ```

- Using the Copy File command. For example, to print the **sys/stat.h** header file, enter the following command:

  ```
  CPYF FROMFILE(QSYSINC/SYS) TOFILE(*PRINT) FROMMBR(STAT)
  ```

Symbolic links to these header files are also provided in directory /QIBM/include.

Errno Values for UNIX-Type Functions

Programs using the UNIX-type functions may receive error information as *errno* values. The possible values returned are listed here in ascending *errno* value sequence.

Name	**Value**	**Text**
EDOM	3001	A domain error occurred in a math function.
ERANGE	3002	A range error occurred.
ETRUNC	3003	Data was truncated on an input, output, or update operation.
ENOTOPEN	3004	File is not open.
ENOTREAD	3005	File is not opened for read operations.
EIO	3006	Input/output error.
ENODEV	3007	No such device.
ERECIO	3008	Cannot get single character for files opened for record I/O.
ENOTWRITE	3009	File is not opened for write operations.
ESTDIN	3010	The stdin stream cannot be opened.
ESTDOUT	3011	The stdout stream cannot be opened.
ESTDERR	3012	The stderr stream cannot be opened.
EBADSEEK	3013	The positioning parameter in fseek is not correct.
EBADNAME	3014	The object name specified is not correct.
EBADMODE	3015	The type variable specified on the open function is not correct.
EBADPOS	3017	The position specifier is not correct.
ENOPOS	3018	There is no record at the specified position.
ENUMMBRS	3019	Attempted to use ftell on multiple members.
ENUMRECS	3020	The current record position is too long for ftell.
EINVAL	3021	The value specified for the argument is not correct.
EBADFUNC	3022	Function parameter in the signal function is not set.
ENOENT	3025	No such path or directory.
ENOREC	3026	Record is not found.
EPERM	3027	The operation is not permitted.
EBADDATA	3028	Message data is not valid.
EBUSY	3029	Resource busy.
EBADOPT	3040	Option specified is not valid.
ENOTUPD	3041	File is not opened for update operations.
ENOTDLT	3042	File is not opened for delete operations.

EPAD	3043	The number of characters written is shorter than the expected record length.
EBADKEYLN	3044	A length that was not valid was specified for the key.
EPUTANDGET	3080	A read operation should not immediately follow a write operation.
EGETANDPUT	3081	A write operation should not immediately follow a read operation.
EIOERROR	3101	A nonrecoverable I/O error occurred.
EIORECERR	3102	A recoverable I/O error occurred.
EACCES	3401	Permission denied.
ENOTDIR	3403	Not a directory.
ENOSPC	3404	No space is available.
EXDEV	3405	Improper link.
EAGAIN	3406	Operation would have caused the process to be suspended.
EWOULDBLOCK	3406	Operation would have caused the process to be suspended.
EINTR	3407	Interrupted function call.
EFAULT	3408	The address used for an argument was not correct.
ETIME	3409	Operation timed out.
ENXIO	3415	No such device or address.
EAPAR	3418	Possible APAR condition or hardware failure.
ERECURSE	3419	Recursive attempt rejected.
EADDRINUSE	3420	Address already in use.
EADDRNOTAVAIL	3421	Address is not available.
EAFNOSUPPORT	3422	The type of socket is not supported in this protocol family.
EALREADY	3423	Operation is already in progress.
ECONNABORTED	3424	Connection ended abnormally.
ECONNREFUSED	3425	A remote host refused an attempted connect operation.
ECONNRESET	3426	A connection with a remote socket was reset by that socket.
EDESTADDRREQ	3427	Operation requires destination address.
EHOSTDOWN	3428	A remote host is not available.
EHOSTUNREACH	3429	A route to the remote host is not available.
EINPROGRESS	3430	Operation in progress.
EISCONN	3431	A connection has already been established.
EMSGSIZE	3432	Message size is out of range.
ENETDOWN	3433	The network currently is not available.
ENETRESET	3434	A socket is connected to a host that is no longer available.

ENETUNREACH	3435	Cannot reach the destination network.
ENOBUFS	3436	There is not enough buffer space for the requested operation.
ENOPROTOOPT	3437	The protocol does not support the specified option.
ENOTCONN	3438	Requested operation requires a connection.
ENOTSOCK	3439	The specified descriptor does not reference a socket.
ENOTSUP	3440	Operation is not supported.
EOPNOTSUPP	3440	Operation is not supported.
EPFNOSUPPORT	3441	The socket protocol family is not supported.
EPROTONOSUPPORT	3442	No protocol of the specified type and domain exists.
EPROTOTYPE	3443	The socket type or protocols are not compatible.
ERCVDERR	3444	An error indication was sent by the peer program.
ESHUTDOWN	3445	Cannot send data after a shutdown.
ESOCKTNOSUPPORT	3446	The specified socket type is not supported.
ETIMEDOUT	3447	A remote host did not respond within the timeout period.
EUNATCH	3448	The protocol required to support the specified address family is not available at this time.
EBADF	3450	Descriptor is not valid.
EMFILE	3452	Too many open files for this process.
ENFILE	3453	Too many open files in the system.
EPIPE	3455	Broken pipe.
ECANCEL	3456	Operation cancelled.
EEXIST	3457	File exists.
EDEADLK	3459	Resource deadlock avoided.
ENOMEM	3460	Storage allocation request failed.
EOWNERTERM	3462	The synchronization object no longer exists because the owner is no longer running.
EDESTROYED	3463	The synchronization object was destroyed, or the object no longer exists.
ETERM	3464	Operation was terminated.
ENOENT1	3465	No such file or directory.
ENOEQFLOG	3466	Object is already linked to a dead directory.
EEMPTYDIR	3467	Directory is empty.
EMLINK	3468	Maximum link count for a file was exceeded.

ESPIPE	3469	Seek request is not supported for object.
ENOSYS	3470	Function not implemented.
EISDIR	3471	Specified target is a directory.
EROFS	3472	Read-only file system.
EUNKNOWN	3474	Unknown system state.
EITERBAD	3475	Iterator is not valid.
EITERSTE	3476	Iterator is in wrong state for operation.
EHRICLSBAD	3477	HRI class is not valid.
EHRICLBAD	3478	HRI subclass is not valid.
EHRITYPBAD	3479	HRI type is not valid.
ENOTAPPL	3480	Data requested is not applicable.
EHRIREQTYP	3481	HRI request type is not valid.
EHRINAMEBAD	3482	HRI resource name is not valid.
EDAMAGE	3484	A damaged object was encountered.
ELOOP	3485	A loop exists in the symbolic links.
ENAMETOOLONG	3486	A path name is too long.
ENOLCK	3487	No locks are available.
ENOTEMPTY	3488	Directory is not empty.
ENOSYSRSC	3489	System resources are not available.
ECONVERT	3490	Conversion error.
E2BIG	3491	Argument list is too long.
EILSEQ	3492	Conversion stopped due to input character that does not belong to the input codeset.
ETYPE	3493	Object type mismatch.
EBADDIR	3494	Attempted to reference a directory that was not found or was destroyed.
EBADOBJ	3495	Attempted to reference an object that was not found, was destroyed, or was damaged.
EIDXINVAL	3496	Data space index used as a directory is not valid.
ESOFTDAMAGE	3497	Object has soft damage.
ENOTENROLL	3498	User is not enrolled in system distribution directory.
EOFFLINE	3499	Object is suspended.
EROOBJ	3500	Object is a read-only object.
EEAHDDSI	3501	Hard damage on extended attribute data space index.
EEASDDSI	3502	Soft damage on extended attribute data space index.
EEAHDDS	3503	Hard damage on extended attribute data space.
EEASDDS	3504	Soft damage on extended attribute data space.
EEADUPRC	3505	Duplicate extended attribute record.

ELOCKED	3506	Area being read from or written to is locked.
EFBIG	3507	Object too large.
EIDRM	3509	The semaphore, shared memory, or message queue identifier is removed from the system.
ENOMSG	3510	The queue does not contain a message of the desired type and (msgflg logically ANDed with IPC_NOWAIT).
EFILECVT	3511	File ID conversion of a directory failed.
EBADFID	3512	A file ID could not be assigned when linking an object to a directory.
ESTALE	3513	File handle was rejected by server.
ESRCH	3515	No such process.
ENOTSIGINIT	3516	Process is not enabled for signals.
ECHILD	3517	No child process.
EBADH	3520	Handle is not valid.
ETOOMANYREFS	3523	The operation would have exceeded the maximum number of references allowed for a descriptor.
ENOTSAFE	3524	Function is not allowed.
EOVERFLOW	3525	Object is too large to process.
EJRNDAMAGE	3526	Journal is damaged.
EJRNINACTIVE	3527	Journal is inactive.
EJRNRCVSPC	3528	Journal space or system storage error.
EJRNRMT	3529	Journal is remote.
ENEWJRNRCV	3530	New journal receiver is needed.
ENEWJRN	3531	New journal is needed.
EJOURNALED	3532	Object already journaled.
EJRNENTTOOLONG	3533	Entry is too large to send.
EDATALINK	3534	Object is a datalink object.
ENOTAVAIL	3535	IASP is not available.
ENOTTY	3536	I/O control operation is not appropriate.
EFBIG2	3540	Attempt to write or truncate file past its sort file size limit.
ETXTBSY	3543	Text file busy.
EASPGRPNOTSET	3544	ASP group not set for thread.
ERESTART	3545	A system call was interrupted and may be restarted.
