Adabas for Linux, UNIX and Windows

XA Support

Version 7.0

October 2022

ADABAS & NATURAL

This document applies to Adabas for Linux, UNIX and Windows Version 7.0 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: ADAOS-XA-70-20220622

Table of Contents

XA Support

This document contains information about XA support.

The following topic is covered:

- *XA Support*

1

About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 XA Support

General

Adabas supports global transactions in accordance with the X/OPEN CAE Specification on Distributed Transaction Processing (XA specification, X/OPEN document number XO/CAE/91/300). Global transactions are typically run under the control of a transaction monitor (in this case the application can use the X/OPEN TX specification interface to the transaction monitor) or an application can use the XA interface directly. Global transactions are normally run across several resource managers (e.g. several Adabas nuclei, which implies multiple Adabas databases).

The XA specification requires that these global transactions are committed via a two-phase commit protocol (2PC). If Adabas acknowledges the commit of a transaction branch in the first phase of 2PC, it must not terminate (commit or roll back) the transaction on its own.

Transactions that are in this first phase are called "pending transactions". If, however, another component that is involved in the transaction fails, such transactions may remain in the pending state for a long time. Pending transactions must "survive" even if the database aborts or is shut down, in which case they are restored by the autorestart of the database.

Utilities that change the contents of the database are, of course, not permitted to run while there are transactions pending. There are, however, situations in which maintaining transactions in the pending state is impractical, for example, when a database is dumped (ADABCK DUMP=*) or a new protection log file is to be opened (ADAOPR FEOF=PLOG), which requires all transactions to be terminated. In such cases, Adabas performs what is called, in the XA specification, a "heuristic termination" of the transaction. This means that Adabas terminates a pending transaction without interacting with the transaction manager (see the messages HEURB and HEURC in the *Messages and Code* section for further information).

Even if Adabas heuristically terminates a transaction, it must not forget about such a transaction unless the transaction manager explicitly allows it to do so. Utilities that change the data in the database can be run while there are no pending transactions, whereas utilities that change the database identification (e.g. ADADBM's NEW_DBID function) cannot be executed when heuristically terminated transactions exist.

The log information of pending transactions may have to be maintained for long periods of time. Because it cannot always be kept in the WORK1 ring buffer, it may have to be copied to a more stable region of WORK1, the so-called XA area (see the WXA parameter of ADAPLP for further information). Pending global transactions are also heuristically terminated if this area overflows.

User-Queue Handling

Each XA session (started by an xa_open call) occupies one user-queue element. This user-queue element is called the XA master user-queue element. In addition, each branch of a global transaction (started by an xa_start call) occupies another user-queue element. This user-queue element is called the XA slave user-queue element. The XA slave user-queue element serves as a pseudo user for the lifetime of a transaction branch, i.e. until a global transaction is finally committed (via xa_commit), rolled back (via xa_rollback) or forgotten (via xa_forget).

Global transactions consist of one or more transaction branches that are identified by transaction identifiers according to the XA specification. This makes it possible that transactions are made of transaction branches from different real-world Adabas users, in contrast to "normal" Adabas transactions that completely belong to a single real-world user. For that reason the master user-queue element can also be seen as the representation of a single real-world user, while the slave user-queue element represents a single or a part of a single global transaction.

Due to the fact that transaction branches may be suspended (calling xa_end with the TM_SUSPEND flag set) and later resumed (calling xa_start with the TM_RESUME or TM_MIGRATE flag set) and further xa_start calls with different transaction identifiers are possible, a master user-queue element may have more than one slave user-queue element at a time. On the other hand a global transaction may consist of more than one user-queue element if the same transaction identifier is used in several calls to xa_start. In that case a xa_prepare call with this transaction identifier brings all user-queue elements into the pending state and a xa_commit or xa_rollback call of the same transaction identifier finally finishes all user-queue elements that together form the global transaction. In that case more than one slave user-queue element may disappear at the same time.

XA master user-queue elements are identified by the user ID `xamaster', the XA slave user-queue elements by the user ID `xaslave'. In addition, XA slave user-queue elements are identified by a login ID that starts with `.X' (where `.' is the NULL character) and where the remaining six bytes of the login ID serve as an internal counter maintained by the XA library, which is incremented with each call to xa_start (see the output of the ADAOPR function DISPLAY=UQ for further information).

The slave and master user-queue elements that belong together all share the same environment specific ID (on UNIX and PC platforms this is the process ID). The master elements are essential for the survival of command IDs following the termination of a global transaction (xa_commit or xa_rollback), which removes the user-queue element generated by the xa_start call.

Slave user-queue elements are subject to timeout restrictions and ADAOPR STOP commands as long as they are not in a pending state. Once they are in a pending state, they remain present for an infinite time, unless they are heuristically terminated, as described earlier in this chapter. Master user-queue elements are not subject to timeout restrictions. They disappear only, if a xa_close call is received or ADAOPR's STOP command is used to remove them (ADAOPR's STOP command only removes a master user-queue element if there are no slaves belonging to it).

If a master user-queue element is stopped, all of the command IDs associated with it will be lost. The next xa_start call, trying to create a new slave user-queue element, will receive a return code of XA_RBTRANSIENT, showing that the master user-queue element for this session is no longer available.

Using Multi-Process Transaction Monitors

A slightly different handling is necessary if the XA library is used by a transaction monitor that consists of more than one process. In that case calls from a real-world user may be sent to Adabas through different work processes of the transaction monitor. It is essential for Adabas to be able to uniquely identify which real-world user has initially supplied a call. In normal operation the name of the originating node (node ID), the login user name (login ID) and the environment specific ID (process ID) of the caller are used to identify the user. This identification is not possible if calls are submitted through different transaction monitor processes (which then implies different process IDs from one real-world user to be used).

To solve that problem a XA user exit must be used. For more information about the creation and definition of a user exit see *User Exits and Hyperexits*.

The XA user exit is identified by the environment variable XAUEX_0 and uses the following interface:

```
#include <adabas.h>
int xauex_0 (char *ubuf)
```

where ubuf is a pointer to a buffer area that receives an eight byte string that uniquely identifies the real world user. A return code of XAUEX_OK signals a successful completion, a return code of XAUEX_IGNORE signals that the result of the user exit is to be ignored. Each other return code is mapped to the XA return code XAER_RMFAIL. xauex_0 is the default function name used for the XA user exit. It is also possible to define a different entry function name according to the section User Exits and Hyperexits.

The XA userexit is called as part of the xa_start call each time a transaction branch is generated. The string that is returned by the user exit overlays the login ID of the user-queue element. So each call to xa_start, using the XAUEX_0 userexit, generates a slave user-queue element showing the returned string as the login ID. The counter that is normally maintained as part of the login ID is in that case maintained in the environment specific ID. The xa_start call also implicitly generates a master user-queue element for this user, using the string returned by the XA userexit as the login ID and using an environment specific ID of zero. Hence, in this environment, there is a master user-queue element for each work process of the transaction monitor, and one for each real-world user. Master and slave user-queue element are again identified through the corresponding `xamaster' or `xaslave' string in the user ID. Those additional master user-queue elements are never removed because no xa_close call is received for them (xa_close is only called for user-queue elements for which xa_open is called at the beginning).

To avoid an increasing number of master user-queue elements filling up the user queue, an additional routine (remove_xa_context) is implemented in the Adabas user interface which is linked with the user application or the transaction monitor. The only parameter for this routine is a pointer to the string making up the login ID. This routine can, for example, be called out of the XA userexit or the Adabas application, to remove master user-queue elements that are no longer to be used (the routine returns a 0 in case of successful removing the user-queue element, and a -1 in case of an error). Additionally ADAOPR's STOP command can be used to remove the master user-queue elements if no slaves are belonging to it.

Nucleus Parameters

If the XA functionality is to be used, the nucleus must be started with the XA option set. If this option is not set, an Adabas response code 230 is returned if an attempt is made to access the nucleus via an XA call. The Adabas XA library maps this return code to the XAER_PROTO return code that is defined in the XA specification.

If the database nucleus is run with XA switched on, it cannot be started for a subsequent session with XA switched off unless there are neither pending transactions nor heuristically completed transactions from the previous nucleus session.

The XA area is allocated in WORK1 XA is switched on. The size of this area is specified by the LPXA parameter of ADANUC.

Each real-world user that supplies XA calls or uses a transaction monitor that supplies XA calls occupies a master user-queue element. In the case of a multi-process transaction monitor one master user-queue element is occupied by each transaction monitor process and one master user-queue element is occupied for each unique identification returned by the XA user exit (if it is not already removed by the remove_xa_context function as described earlier in this chapter). Additionally, in all cases, each transaction branch that is not already committed or rolled back occupies a slave user-queue element. Each pending or heuristically terminated transactions occupies at least one user-queue element, which means that the nucleus parameter NU (number of user-queue elements) should be increased when the nucleus is run with XA switched on.

Applications Using XA Functionality

The XA specification requires that Adabas publishes the following information (refer to the XA specification, section 7.2 for further details):

- The XA specification defines the structure xa_switch_t, which gives entry point and other information to the transaction manager. The name of the corresponding Adabas structure is adaxasw.

- The text string within the resource-manager switch that specifies the name of the resource manager is "ADABAS".
- The xa_open call accepts an information string of the form “dbid=<dbid>”, where <dbid> should be replaced by the database identifier of the Adabas database to be used. The xa_close call accepts an empty string.
- There are differences to non-DTP (distributed transaction processing) operations:
  - ET data cannot be used, because there is no way that they can be passed to Adabas via the xa_commit call.
  - The multi-fetch option for ET/BT is not supported.

The operating system entity that is considered to be a thread of control is an operating system process. The Adabas XA interface does not support any use of multiple threading facilities of the underlying operating system.

The XA specification leaves some implementation choices open to whoever implements the resource manager. The following choices have been made for Adabas:

- The read-only optimization has been implemented, i.e. if an xa_prepare is called for a transaction that has not issued database update commands (leaving the slave user-queue elements in ET state), the transaction will be committed immediately (without a separate call to xa_commit or xa_rollback) and the response XA_RDONLY is returned.
- Association migration is supported. This means that a suspended association (created by xa_end with the TMSUSPEND and TMMIGRATE flag being set) can be resumed in a thread of control other than the one in which the suspension occurs (in this case xa_start must be called with the TMRESUME flag set).
- Adabas does not dynamically register with the transaction manager, i.e. the transaction manager has to call xa_start for each global transaction.
- The asynchronous call mode is not supported, i.e. all XA calls are blocking.
- Heuristic completion is used as described earlier.
- Adabas does not support the xa_start_2 call.

The XA functionality is only available via the Adabas XA interface, which means that there are no Adabas direct calls to start or end a global transaction. The XA interface does, however, cooperate with the existing Adabas direct calls. Direct calls which affect the Adabas transaction logic (BT, CL, ET, OP) will be rejected if a global transaction is active (response code 230).