Adabas Transaction Manager

Adabas Transaction Manager Programmers Guide

Version 8.2.2

April 2020

ADABAS & NATURAL

This document applies to Adabas Transaction Manager Version 8.2.2 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: ATM-PGGUIDE-822-20211207

Table of Contents

Preface v
1 About this Documentation 1
    Document Conventions 2
    Online Information and Support 2
    Data Protection 3
2 Programming transactions in Natural and 3GL 5
    Programming Adabas transactions in Natural 6
    Programming 3GL Adabas transactions 6
3 Well-formed distributed transaction enforcement 7
4 Automatic distributed transaction integrity 9
5 Detection of ill-formed distributed transactions 11
6 Continuous operation 13
7 Adabas ADARUN DTP= settings 15
8 Distributed transaction time limit 17
9 ET data and ET identity processing 19
    Using ET identities (ETID) 20
    ET Data 20
    Continuous operation mode and ET Data 21
    External transaction coordinators and ET Data 21
    Multi-system dynamic transaction routing 22
10 Other considerations 23
    Extended Hold Processing 24
    Shared Hold Status 24
    Close Commands 25
    Open Commands 25
11 Pending Response Codes 27
12 Open distributed transaction processing 29
    Using CICS RMI 30
    Using RRMS 31
13 Triggers 33

Preface

This section provides information that will help programmers in an Adabas Transaction Manager environment.

Note: Throughout this section the processing described is equivalent for ET versus BT unless specifically stated to be different.

The following topics are provided:

Programming transactions in Natural and 3GL
Well-formed distributed transaction enforcement
Automatic distributed transaction integrity
Detection of ill-formed distributed transactions
Continuous operation
Adabas ADARUN DTP= settings
Distributed transaction time limit
ET data and ET identity processing
Other considerations
Pending response codes
Open distributed transaction processing
Triggers

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

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

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **https://documentation.softwareag.com**.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.aspx** and give us a call.

Software AG Tech Community

You can find documentation and other technical information on the Software AG Tech Community website at **https://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

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 Programming transactions in Natural and 3GL

- Programming Adabas transactions in Natural 6
- Programming 3GL Adabas transactions 6

Programming Adabas transactions in Natural

Here's a simple Adabas transaction:

1.	Modify data in Adabas database A
2.	Issue END TRANSACTION

This is indeed extremely simple. However, there is only one database that is modified so let's see what happens when multiple databases are involved:

1.	Modify data in Adabas database A
2.	Modify data in Adabas database B
3.	Issue END TRANSACTION

Programming 3GL Adabas transactions

Here's how the previous logic looks when it is implemented in 3GL:

1.	Modify data in Adabas database A
2.	Modify data in Adabas database B
3.	Issue ET command to Adabas database A
4.	Issue ET command to Adabas database B

The difference between programming transactions in Natural versus 3GL is that Natural makes it very simple (as it is supposed to do). In Natural you code a single `END TRANSACTION` statement no matter how many databases are modified. In 3GL programming you code specific `ET` commands for each of the modified databases.

This difference isn't difficult to understand, and appears to make sense. However, it highlights an extremely important area in transaction programming that we call ***well-formed distributed transactions***.

3 Well-formed distributed transaction enforcement

A transaction is considered to be *distributed* when it modifies more than one database (RM) in the same transaction. This means the Adabas command interface has to be used carefully to ensure compliance with correct distributed transaction behavior. In other words, a well-formed distributed transaction is one where the modifications to all the databases are all applied (committed) together - and it is the programmer's responsibility to make sure this happens.

Natural ensures that distributed transactions are well-formed because it automatically generates a series of `ET` commands for all modified databases when it encounters an `END TRANSACTION` statement. There is no programmer control over this, nor should there be. Natural keeps a list of all modified databases internally so when an `END TRANSACTION` is encountered it simply iterates through the list issuing `ET` commands. Therefore, Natural truly enforces the use of well-formed distributed transactions

None of what Natural does guarantees integrity of distributed transactions, but it does ensure that where they are used they are well-formed. To guarantee integrity you need to adopt Adabas Transaction Manager, see **Automatic distributed transaction integrity**.

Of course, 3GL programming is not as easy as Natural. The following 3GL logic is illegal in terms of distributed transaction programming because it applies a modification to one Adabas database but undoes the modifications to the other - this is poor (ill-formed) distributed transaction programming:

1.	Modify data in Adabas database A
2.	Modify data in Adabas database B
3.	Issue ET command to Adabas database A
4.	Issue BT command to Adabas database B

4 Automatic distributed transaction integrity

Whether Natural generates multiple `ET` commands or if they are programmed in 3GL, the fact is that a series of `ET` commands for multiple databases does not provide guaranteed distributed transaction integrity because if there is a system crash, for example, between one `ET` and the next an inconsistency occurs.

Adabas Transaction Manager guarantees the integrity of distributed transactions by reacting to the first ET of the series to make sure the full distributed transaction is applied across all the modified databases. As follows:

1.	Modify data in Adabas database A
2.	Modify data in Adabas database B
3.	Issue ET command to Adabas database A ■ ATM transparently performs two-phase commit for all modified databases
4.	Issue ET command to Adabas database B ■ This ET now appears redundant – but it is not, see **Detection of ill-formed distributed transactions**.

5 Detection of ill-formed distributed transactions

Adabas Transaction Manager applies distributed transaction integrity as soon as the first `ET` of a series is encountered. This suggests the subsequent `ET` commands of the series are not needed but that is incorrect. Well-formed distributed transaction programming is still necessary and enforced. Here's why:

1.	Modify data in Adabas database A
2.	Modify data in Adabas database B
3.	Issue ET command to Adabas database A ■ ATM transparently performs two-phase commit for all modified databases
4.	Issue BT command to Adabas database B ■ **The application is trying to execute an illegal operation!**

In the example above, the application is breaking the rules of distributed transaction processing by adopting unsafe behavior. The rules of DTP stipulate the modifications for all databases in a distributed transaction are all applied or all undone; it is not legal to have a mixture of some applied and some undone.

In this situation Adabas Transaction Manager throws an error when the `BT` in point 4 above is encountered (response 240 sub-code 496). Here's another example, just a little more complicated:

1.	Modify data in Adabas database A
2.	Modify data in Adabas database B
3.	Issue ET command to Adabas database A ■ ATM transparently performs two-phase commit for all modified databases

4.	Modify data in any database (A or B or another) ■ **The application is trying to execute an illegal operation!**

In this example, the application issues only one ET command, it does not issue an ET command to the second database. Then it goes on to do other work – in this case to start doing modifications in a new transaction. This logic is breaking the rules of distributed transaction processing with unsafe behavior too. In this situation Adabas Transaction Manager throws an error when the modification in point 4 above is encountered (response 240 sub-code 496). Some people find it difficult to understand why this example breaks the rules so consider what might happen without Adabas Transaction Manager being used:

1.	Modify data in Adabas database A
2.	Modify data in Adabas database B
3.	Issue ET command to Adabas database A ■ Adabas database B backs out the changes made so far due to resources shortages or some other reason.
4.	Modify more data in Adabas database B
5.	Issue ET command to Adabas database B

This is really not a good situation when Adabas Transaction Manager is not present. Here is what has happened…

1.	The modification to Adabas A is applied.
2.	The first modification to Adabas B is undone.
3.	The second modification to Adabas B is applied… ■ **Only half of the modifications to Adabas B are applied.**

The only way to be sure distributed transactions are executed correctly in your business systems is to make sure there is no room for doubt by

1. adopting Adabas Transaction Manager

   and

2. allowing it to enforce well-formed distributed transaction processing behavior, as it does by default.

6 Continuous operation

From time to time one or more components of Adabas Transaction Manager may suffer transient outage. The client runtime control Continuous operation mode defines the level of tolerance to be adopted in these circumstances. The default behavior is to avoid disruption and allowing as much uninterrupted continuous operation as possible. However some companies may choose to disallow (reject with an error) distributed transactions during outage by altering the tolerance level for continuous operation for some or all clients.

The default mode (FORCE) allows Adabas Transaction Manager to temporarily generate series of `ET` commands instead of performing full DTP coordination until the outage ends. When the outage is over then full DTP coordination resumes automatically (on the next new transaction for each client session).

7 Adabas ADARUN DTP= settings

When adopting Adabas Transaction Manager it is recommended to run Adabas with DTP=RM. This setting enables the RM role of Adabas so that it can be coordinated by Adabas Transaction Manager to provide full DTP support. These databases are referred to as Adabas RMs.

DTP=NO can be used to disable the RM role of Adabas. This is not recommended. When this setting is used the database cannot participate in DTP coordination so all transactions for this database are processed using serial ET, with no guarantee of DTP integrity.

8 Distributed transaction time limit

In the same way that Adabas provides a time limit for its local database transactions, Adabas Transaction Manager supports a time limit for distributed transactions. When the time limit expires for a distributed transaction Adabas Transaction Manager attempts to complete it by applying or undoing it, depending on its status. If the transaction is undone, the application receives response code 9 with a suitable sub-code as applicable.

It is recommended that the distributed transaction time limit is lower than all of the individual Adabas transaction time limits, thereby enabling Adabas Transaction Manager to dictate when timeouts occur or not, which is what it is intended to do.

9 ET data and ET identity processing

- Using ET identities (ETID) 20
- ET Data 20
- Continuous operation mode and ET Data 21
- External transaction coordinators and ET Data 21
- Multi-system dynamic transaction routing 22

Using ET identities (ETID)

It is possible for an application to use an ETID for some database sessions, and concurrently to use other database sessions with no ETID. It is also possible for a client to use different ETIDs concurrently in different database sessions; however, Software AG strongly recommends that the same ETID is used for all databases that are used.

ET Data

ET data is a special type of application data. Changes to ET data take effect if and only if a transaction completes successfully (is applied) with an ET command (or CL).

Adabas Transaction Manager supports the use of ET data with distributed transactions. If no new ET data is used when the transaction is applied the existing ET data remains unchanged, similar to how standard Adabas handles ET data.

When an application issues an ET command with ET data when no valid ETID has been provided, Adabas Transaction Manager, like Adabas will not store the ET data persistently. Instead, it will proceed as follows:

- If the target database of the ET command is being managed using the two-phase protocol, the ET data will be ignored, but Adabas Transaction Manager will not draw attention to this fact by means of a non-zero response code;
- Otherwise, the ET data will be passed to the database which the application specified on the ET command, and to no other database.

When ET data is stored by an ET or CL command for a session using a valid ETID, its location is determined by the setting of the client runtime control ET data storage location.

Note: It is important that the same mechanism and store location is used from one execution to the next otherwise ET data will be lost. For example, if the application is configured to store ET data in a database one day, and then changed to use the TM recovery file the next, then it is likely that ET data will be lost during this changeover – unless special action has taken place to make sure the ET data has been moved from the old store to the new store.

Software AG strongly recommends applications should use the same ETID in all databases for one client. When this recommendation is followed, there can be no confusion about which ETID should be used when ET data is to be stored or read.

For information about:

- controlling the storage of ET data, see the ET data storage location client runtime control and ET data storage location TM control.

- establishing the current ET data of your applications in the TM recovery file before they execute, see ET Data Management.

Continuous operation mode and ET Data

When a client session is temporarily running in continuous operation mode, ET data requests are directed to the database specified in the Adabas control block even though the client is configured to use the TM recovery file. This could result in

- incorrect ET data being read.
- ET data being written to a database from which Adabas Transaction Manager is not able to retrieve it.
- incorrect results when Adabas Transaction Manager attempts to recover a transaction on behalf of the user

For these reasons, it is strongly recommended that the client control *Continuous operation mode* is set to NO when ET data is configured to be maintained in the TM recovery file.

External transaction coordinators and ET Data

When running with the CICS Syncpoint Manager or Recoverable Resource Management Services (RRMS), it is not possible to synchronize the storage of ET data when an unsolicited syncpoint occurs, because CICS and RRMS syncpoints have no knowledge of ET data.

If an application stores ET data and runs in a CICS/RMI or RRMS environment, you can ensure that the storing of its ET data is synchronized with the two-phase commit process by conforming to the following rules:

- Any syncpoint for which ET data is to be stored must be triggered by an ET or CL command.
- The ET or CL command that triggers the syncpoint must also supply the ET data; that is, if the application issues a series of ET commands to different databases, the first ET must supply the ET data.

In an IMS TM system whose transactions are coordinated by RRMS, it is not possible to store ET data synchronously with an RRMS syncpoint. IMS allows an RRMS commit syncpoint to take place only at the successful completion of message processing, and this syncpoint cannot be triggered by an ET or CL command.

Multi-system dynamic transaction routing

Do not use the TM option for defining the storage location of ET data for clients operating in a multi system dynamic transaction routing environment.

Refer to the TM control ET data storage location and Client Runtime Control ET data storage location for more information on defining the storage location of ET data.

10 Other considerations

- Extended Hold Processing 24
- Shared Hold Status 24
- Close Commands 25
- Open Commands 25

Extended Hold Processing

The P and M command options of ET and BT commands allow a program to keep some records in hold status even after a transaction is applied or undone. Adabas Transaction Manager supports these options when the *Extended Hold* client runtime control is set. Extended hold processing proceeds as follows:

- At the point where the distributed transaction is applied or undone, the P/M option on the first ET/BT is honored (if used). In addition, all records in the other databases remain in hold until the ET for each database in turn is encountered.
- As each ET (or BT) in series is encountered the P/M options are honored for each database in turn.
- The consequence of this unavoidable processing is that some or all records are held longer than they need to be, and additional commands are issued in order to resolve the hold processing to obey the wishes of the application.

If the extended hold option is not active (the default setting), any ET or BT command will cause all the user's held ISNs to be released in databases where changes have been made, except possibly in the database that is the target of the command; for this database, any P or M option present in the command will be honored. This is by far the most efficient mode of execution.

Note: Extended hold processing is only relevant when transaction control is exerted by ET or BT command.

Shared Hold Status

The H option of command option 3 for an ET or BT ACBX call allows a program to keep a record in shared hold status indefinitely (until the next ET or BT command). Adabas Transaction Manager supports this option as follows:

- At the point where the distributed transaction is applied or undone, the H option on the first ET/BT is honored.
- As each ET (or BT) in series is encountered, if a shared hold status conflict is detected (for example, a "H" option is subsequently specified for a database that was involved in the earlier distributed transaction) then a response 240 sub-code 588 will be returned to the calling program. This has no effect on the transaction outcome, which was determined earlier, but is an indication to the program that it should review its use of this option.

Note: Adabas does not currently support the shared hold status "H" option for the internal Yx commands used by Adabas Transaction Manager.

Close Commands

Since a close command implies end-of-transaction, every CL command triggers the same processing as for an ET command, except that:

- the user is also closed in the target database;
- if the target database is at ET status, the current distributed transaction is not affected.

Open Commands

An OP command sent to a database in which the client has no uncommitted changes or held records will simply be passed to its target without affecting the client's distributed transaction status. This logic allows "open on demand" processing without interference in distributed transaction processing.

An OP command sent to a database that is actively involved in the current distributed transaction causes the distributed transaction to be wholly undone.

If the OP command specifies that ET data is to be read, the above processing occurs. If processing is successful, the user's ET data is returned to the calling program. This applies even if the target database has the parameter setting DTP=NO. The ET data is read either from the ATM transaction manager's recovery file or, if the transaction manager is running with the parameter setting TMETDATA=TARGETS, from the database indicated by the OP command.

11 Pending Response Codes

Sometimes a transaction is terminated in such a way that the owner should receive a non-zero response code, but in circumstances in which it is not possible to return a response code; for example, the transaction manager backs the transaction out because its distributed transaction time limit has been exceeded. When this happens, the manager stores details of the pending response code in a list, and returns it to the client at the first possible opportunity. Pending response codes can be listed and displayed using Adabas Transaction Manager's online administration tool.

As with standard usage of Adabas, the time for which a pending response code is preserved depends on whether ETIDs are used.

If the transaction which caused the pending response code involved no RM database sessions with ETIDs, the pending response code is discarded as soon as any of the following conditions has been satisfied:

- The response code has been returned to the client.
- The client is known to have disappeared.
- All Adabas RMs that took part in the transaction have been restarted.
- The transaction manager terminates.

If the transaction which caused the pending response code involved one or more RM database sessions with ETIDs, the pending response code is discarded as soon as any of the following conditions has been satisfied:

- The response code has been returned to the client.
- The client session is known to have disappeared.
- All Adabas RMs that took part in the transaction, and for which an ETID was in use, have been restarted.

Note: In this case a pending response code will survive a restart of the transaction manager.

12 Open distributed transaction processing

- Using CICS RMI 30
- Using RRMS 31

Adabas Transaction Manager allows open transactions so that changes to Adabas databases can also be coordinated alongside transactions with other vendors (example DB2, etc). You must set *Open distributed transaction support* and related settings accordingly (such as *Adabas transaction dynamics*) to use either CICS/RMI or RRMS.

Open transactions also means you must consider the effect of control over transactions being exerted by more than the usual `ET` and `BT` commands. The default for Adabas is that transactions are able to span TP system message-pairs. However, this is restricted when running with other vendor transactions. Transactions are not able to span message-pairs. Basically, open transaction programming demands that screen interactions force a transaction boundary where any in-flight transaction is automatically committed. This can be a severe limitation on Adabas applications that have not been designed to work with such restrictions.

Many Adabas applications are not able to accommodate these restrictions. A number of settings exist to help try to alleviate the difficulties in the *Open distributed transaction support* area. Essentially, the design of the application is governed by the open transaction model that is used.

Using CICS RMI

- APPC Applications with CICS RMI
- Continuous operation mode with CICS RMI

Adabas Transaction Manager enables inter-operation with IBM's CICS Resource Manager Interface (CICS RMI). This enables CICS applications to execute transactions that distribute across Adabas and any other vendor offerings that also work with CICS RMI (such as DB2, VSAM, etc).

CICS RMI must be configured correctly according to IBM documentation along with appropriate settings in the area of *Open distributed transaction support*. Correct configuration will enable automatic resynchronization as well as open transaction coordination interactions.

From a programming viewpoint transactions can be applied (committed) by the usual Adabas commands (`ET/CL`) or EXEC CICS SYNCPOINT in the application. You can set client runtime controls to specify whether you want some or all of these to be allowed. However, if there has been neither of these when a (pseudo-conversational) screen interaction takes place (or end of CICS task) then CICS RMI will itself enforce a unilateral commit to take place.

APPC Applications with CICS RMI

Some specialized applications are designed to use Advanced Peer-to-Peer Communication (APPC) in CICS RMI. APPC is a protocol for two programs to work together as peers, usually across different CICS (or compatible) TP systems in the same or different operating systems.

The APPC protocol demands the programmer is fully trained to implement peer programs that dynamically negotiate the explicit and implicit changing peer roles for control of transactions (this is governed by the rules of CICS RMI APPC not by Adabas Transaction Manager). For example, when one peer issues `SYNCPOINT ROLLBACK` either explicitly or by issuing a `BT` command the rules of APPC demand that it cannot then issue an `ET` (or `SYNCPOINT`)…this is all related to implied/explicit role play within the APPC protocol. When these protocol rules are broken CICS RMI will issue ASP2 abends (etc). These abends are not an indication of problems in Adabas or Adabas Transaction Manager; they are indication of the APPC programming protocol not being adhered to – in other words application program bugs. However, as stated the APPC protocol is very complex so it can be very difficult to solve such application problems.

Continuous operation mode with CICS RMI

Continuous operation mode can be used with CICS RMI, and when it is in play during outage Adabas transactions are managed with series of `ET/BT` commands as usual. And also as usual, when the outage is ended full DTP mode resumes automatically. If continuous operation mode takes effect in the middle of transaction completion the effects can vary according to the point at which it happens. This may mean the transaction is undone or is deferred or even a CICS transaction abend may occur (something CICS RMI often does unilaterally when symptoms occur during transaction completion). New transaction completions will follow the settings you choose for continuous operation until the outage ends and normal operation resumes.

Using RRMS

- Using RRMS with IMS TM

Adabas Transaction Manager enables inter-operation with IBM's Recoverable Resource Management Services (RRMS). This enables any combination of resource managers (typically DBMSs) to participate in distributed transactions that are coordinated by Resource Recovery Services (RRS), a component of RRMS that drives the two-phase commit protocol.

RRMS must be configured correctly according to IBM documentation along with appropriate settings in the area of *Open distributed transaction support*. Correct configuration will enable automatic resynchronization as well as open transaction coordination interactions.

From a programming viewpoint transactions can be applied (committed) by the usual Adabas commands (`ET/CL`) or by a system call to RRMS. You can set client runtime controls to specify whether you want some or all of these to be allowed.

RRMS support is provided for single-user, single-TCB batch applications, and for applications running under IMS TM.

Using RRMS with IMS TM

If your applications run under IMS TM, and IMS allows its transactions to be coordinated by RRMS, Adabas Transaction Manager can ensure that their Adabas changes are committed (or backed out) in a synchronized manner, under the control of RRMS. An IMS commit syncpoint causes RRMS to carry out a commit operation for all changed resources that are managed by RRMS-enabled resource managers. An IMS rollback syncpoint causes all changes to be backed out.

The completion of an IMS message (normally this means screen I/O) causes a syncpoint to take place. In the case of successful completion, this is a commit syncpoint. Moreover, a commit syncpoint implies the completion of processing for the current message. For this reason, an `ET` or `CL` command that is issued during the processing of a message will not cause an RRMS commit syncpoint; any pending Adabas changes will be committed, but non-Adabas resources will be unaffected. Further, any held ISNs will be released, or will remain in held status, depending on the presence of `P` or `M` command options, and the setting of the Extended Hold client runtime control.

13 Triggers

Triggers can execute distributed transactions under the control of Adabas Transaction Manager. Refer to the documentation for the Adabas System Coordinator to find out how to configure the System Coordinator for a trigger environment.

If your application causes participating triggers to be fired, and you require a distributed transaction to include changes made by both the application program and the participating trigger, you must ensure that the very first change command (store, delete or update) of the distributed transaction is issued by the application program, not by the trigger. This is necessary because the originating client must be the root of the distributed transaction, not the transient proxy trigger.