

## **SQLBase 8.0 and COM+ Transactions**

---

**By Suren Behari  
Product Manager**

## TABLE OF CONTENTS

<b>Abstract</b> .....	<b>3</b>
<b>Introduction</b> .....	<b>3</b>
<b>Microsoft Transaction Server (MTS) Support</b> .....	<b>3</b>
What is MTS?.....	3
MTS architecture.....	4
Distributed Transactions .....	5
Components of Distributed Transactions with SQLBase 8.0.....	6
Implementation of a Distributed Transaction .....	7
Summary .....	8
<b>Setting Up the MTS Sample Bank Application</b> .....	<b>8</b>
<b>Download Source Code</b> .....	<b>10</b>
<b>Requirements</b> .....	<b>10</b>

## **Abstract**

This paper discusses Microsoft Transaction Server (MTS) and how SQLBase 8.0 supports COM+ transactions (also known as MTS). Samples of COM+ applications are also discussed.

## **Introduction**

The boundaries of a transaction are not limited by a database anymore. Transactions cover complex business processes that include updating databases from different vendors and at different sites. SQLBase COM+ Transaction support allows to fully integrate SQLBase into COM+ transactions including full commit and rollback support for overall transactions. Many programming languages including Team Developer can be used to create COM+ transactions. For example one combined transaction can move money from a local bank account in SQLBase to a central account in Microsoft SQL Server. The entire transaction either succeeds or if fails is rolled back in all participating databases.

## **Microsoft Transaction Server (MTS) Support**

To be able to develop MTS applications against SQLBase 8.0 using Gupta's Team Developer, or any of the other development language like Visual C++, Visual Basic, Delphi, Power Builder, we must first discuss and understand some concepts including:

- Microsoft Transaction Server
  - MTS Architecture
- Distributed Transactions
  - The ACID Properties
- Components of Distributed Transactions with SQLBase 8.0
  - The DTC
  - Two Phase Commit
  - SQLBrm
  - Communication Threads
  - Recovery
  - Logging
- Implementation of a Distributed Transaction

## **What is MTS?**

MTS is a distributed runtime environment for COM objects that provides an infrastructure for running objects across a network. MTS is a combined object request broker (ORB), resource manager, and transaction monitor.

MTS provides automatic transaction management, database connection pooling, process isolation, automatic thread pooling, automatic object instance management, resource sharing, role-based security, transaction monitoring within distributed applications, and more.

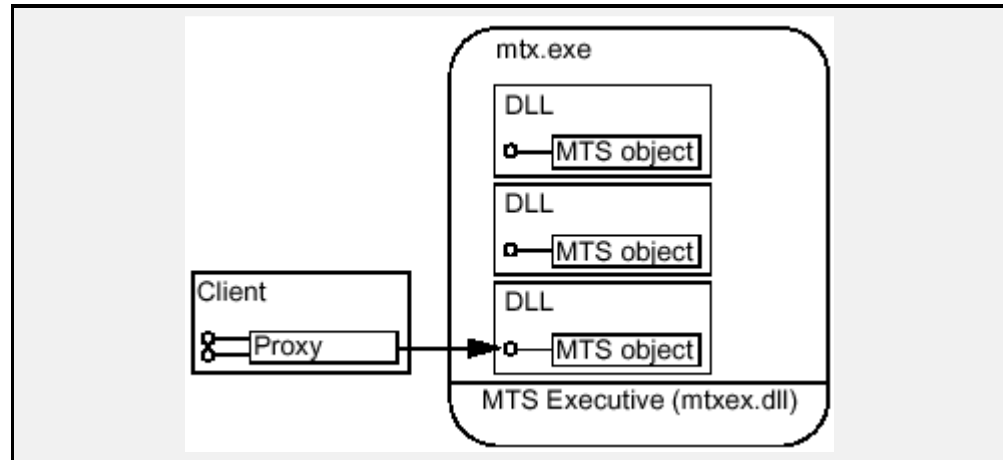
These services are necessary for scaling server-side components and supporting a substantial number of concurrent client requests. MTS performs all of these services automatically, and without the need for application developers to write special code.

**MTS is a combined object request broker (ORB), resource manager, and transaction monitor**

A developer can therefore develop server-side components that behave as if only a single client is connected at a time.

### MTS architecture

MTS is a DLL surrogate process. DLL server components are loaded into the MTS surrogate (MTSx.exe) process along with the MTS Executive (MTSxex.dll) as shown below.



You can use Team Developer to build an MTS object as a DLL. MTS creates proxies for the client. To manage every single instance of a component installed under its control, MTS creates a sibling object called a context object whose interface is exposed by `IObjectContext`.

You use `IObjectContext` to commit or abort a component's transaction, to enable or disable transactions, and to check the security of a component's caller. This context contains information about the object's execution environment, such as the identity of the object's creator and, optionally, the transaction encompassing the work of the object.

An MTS object and its associated context object have corresponding lifetimes. MTS creates the context before it creates the MTS object. MTS destroys the context after it destroys the MTS object.

MTS coordinates the communication between components and databases over a set of pre-established connections (connection pooling).

MTS clients use COM to create MTS components and MTS manages execution of the components. Clients can receive MTS events and get complete/rollback status.

An activity is a collection of MTS objects that has a single distributed thread of logical execution. Each MTS object belongs to a single activity.

COM+(the combination of COM with a new version of MTS in Windows 2000) will support these features:

- *Deferred activation*, where MTS defers physically creating an object until the client actually calls one of its methods.
- *Early deactivation*, where MTS deactivates an object (after a period of inactivity) even while the client still holds a reference to the object. Later, when the client calls a method, MTS creates the object again.

## Distributed Transactions

An example of a transaction would be a database update, where a series of events occurred which led to some data in the database to update its value. To make this change permanent we must commit the update, otherwise if we wish to reverse our change we must rollback the update. The transaction at this stage is complete. This process can be thought of as a single transaction.

If, however, there were multiple databases that need to be updated in a networked environment then this transaction would be referred to as a distributed transaction.

Both Single and Distributed transactions are defined by the properties known as the ACID properties this will be discussed next.

## The ACID Properties

ACID is an acronym for the four properties that defines a transaction, these include **A**tomic, **C**onsistent, **I**solated and **D**urable, and these will be defined now.

Let's use the example that in order to update customer information, we may need to updates three tables, customer, address and contact.

### Atomic – all or none

**Atomic** means all the series of events for the transaction are done wholly or completely or none of the work for the series of events are done.

From the example, if we cannot update something in the address table but we can in the other two tables then the transaction must rollback. Only if in all three tables we were able to update the data successfully then we will commit the transaction.

### Consistent – preserve that data until transaction completed

The term **consistent** means, that we must preserve the state of the data, until the transaction has completed as a whole.

From the example, the transaction must have kept a record of the previous values for all tables so that in case there is a failure in the transaction, the data can be reverted back to its original values.

### Isolated – no dependant on other transactions

**Isolated** refers to transactions should run as though they were the only transaction running and should not depend on other transactions.

From the example assuming that the update of all three tables was successful and also assuming that the records being updated were not trying to be updated by someone else then this transaction should neither interfere nor rely on any other transaction to complete.

### Durable – commit or rollback dependent on unanimous success

**Durable** is the fact that if all events have completed successfully then the transaction will definitely commit and if even one event failed then the transaction will rollback.

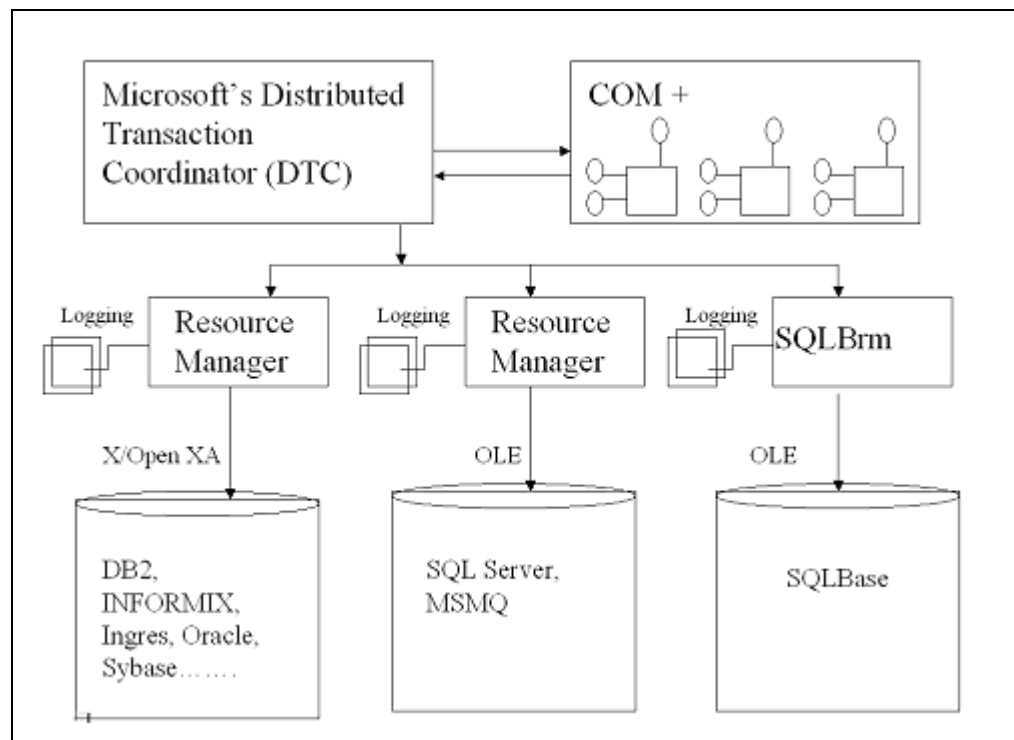
From the example, if all events (rows in the tables) were updated without failure then a commit should be performed to make the change permanent in the

database. On the reverse side of this if one of the updates failed then the entire transaction should be rolled back so the data has its original values.

### Components of Distributed Transactions with SQLBase 8.0

As previously stated, in a distributed transaction all series of events that are part of that transaction must have completed successfully for the entire transaction to be committed.

The diagram below shows the active parts of an implementation of the distributed transaction environment. The components that need to be defined and discussed in this section include the Distributed Transaction Coordinator (DTC), the SQLBrm (SQLBase Resource Manager) and the use of OLE DB between the SQLBrm and the SQLBase database.



### Distributed Transaction Coordinator (DTC)

Microsoft provides a tool called the Distributed Transaction Coordinator (DTC), which is part of the Component Services in the Administrative Tools menu in Windows 2000.

The purpose of this is to coordinate the components that are involved in a transaction and it acts as a transaction manager for each computer that is involved in the transaction.

The main task of a transaction manager is to know which events or components are part of the transaction and also to provide a unique identification for that transaction, so that it can be shared with the components of the transaction. The transaction manager also coordinates the transactions' outcome.

Upon successful completion of each event and the transaction entire, the transaction manager will perform what is called a two-phase commit and this will be explained next.

### **Two-Phase Commit**

The two-phase commit can be described as:

Phase 1: The transaction manager requests a component status from each individual component involved in the transaction on whether they can commit or whether they are going to abort.

Dependant on the component status, phase 2 can result in one of the two options listed

Phase 2 (scenario 1): If the unanimous response from all the components involved in the transaction is 'commit' the transaction manager will then commit the each component and therefore the entire transaction will be committed.

Or

Phase 2 (scenario 2): If one or more components involved in the transaction returns a response component status of abort then the transaction manager will abort the entire transaction which may involve rolling back any database involvements.

### **SQLBrm – SQLBase Resource Manager**

As the name suggests, the SQLBrm is responsible for managing the resource - Gupta SQLBase database. A resource manager is responsible for managing a resource that is can be reversed or recovered and the resource must be able to work in a transactional manner and support the ACID properties as previously described.

The SQLBrm is responsible for managing communication threads by either 'waking' a 'sleeping' thread (using a thread that already been created but is currently not in use) or creating a new thread. This thread maybe used either to connect a server communication thread to a session or a client communication thread to a session.

SQLBrm is also responsible for handling the actual commit or abort transaction with the database. The DTC will get the status from the components of a transaction and if the status is to commit or abort as described above, it is the DTC will coordinate with SQLBrm and SQLBrm will do the actual commit or abort against the SQLBase database.

SQLBrm also handles client and server communication threads by enlisting them when they are required and de-enlisting them when they are no longer required.

If there is a shutdown in SQLBase, SQLBrm will handle the shutdown of the communication threads, it will shut down the listener, halt any active communication threads and stop all sessions.

### **Implementation of a Distributed Transaction**

Either the SQLBase OLE DB Provider or ODBC driver will connect, as normally, to SQLBase, but through SQLBrm.

The contact by the OLE DB Provider or ODBC driver will cause the Listener to create a session and request the Manager for a client and a server thread to be attached to this session.

The Manager will either provide an existing, not attached, client and server thread. If there isn't a free communication thread, then a new one is created.

The client is notified of the acceptance and sends a connect request. The request is passed from the client thread to the server thread into SQLBase and the response is sent back from SQLBase to the server thread, the client thread, and back to the client.

If the client is using an MTS transaction, SQLBrm will be instructed to enlist in the transaction. SQLBrm marks the session as enlisted.

Any database activity follows the same path as the connect request.

When the client commits the MTS transaction, the DTC informs all enlisted resource managers to Prepare for a commit, then commit. SQLBrm marks the session as committed and deletes any logged data. When the client drops the connection, that is passed to SQLBase and the client thread is un-attached from the session.

When SQLBase acknowledges the drop, the server thread is un-attached and the session is deleted.

Note that an MTS transaction may extend past several connections or several MTS transactions and may be part of a single connection.

This is different than the usual SQLBase concepts of a transaction lasting from connect to disconnect.

### **Summary**

We should now have an understanding of a distributed transaction and the components that make an implementation of a distributed transaction with Gupta SQLBase and SQLBrm. We should also understand the logging that both SQLBrm and DTC do in order to recover transactions during a system failure.

So now let's learn how to practically set up the environment and then create an example of how to use Gupta's Team Developer to create MTS Applications against SQLBase 8.0

### **Setting Up the MTS Sample Bank Application**

The Sample Bank application is a banking services application that creates the infrastructure to support the program on the island database, creates new accounts and deletes existing ones, opens and closes accounts, allows deposits and withdrawals and also allows you to view your balance.

Running the Sample Bank application allows you to test your installation of MTS with Gupta SQLBase 8.0 as well as practice package deployment and administration.

Sample Bank has components are written in Team Developer and the source code is attached.

### **Build the Components (Team Developer)**

Currently you can build the COM+ components in Team Developer.

To build the Team Developer components open the MTS\_TD\_COM\_with\_OLEDB.app in Team Developer 2.1.

In the 'Build Settings' menu option select the 'Build Target' tab and then name the DLL as MTS\_TD.dll then select 'MTS COM Server (DLL)' radio button, in the COM Server tab select the 'Single Threaded Apartment', ensure the version major is 1 and minor is 0.

Now build the COM.

### **Build the client program (Team Developer)**

The sample client program that drives the components is written in Team Developer and is called 'MTS\_TD\_Client\_with\_OLEDB.app'

Using the ActiveX Explorer menu option, select the COM called 'MTS\_TD 1.0 Type Library' and generate the APL for all CoClasses and Interfaces. Now compile and save the APP.

Build the executable 'MTS\_TD\_Client\_with\_OLEDB.exe'

### **Create a MMC Console**

Go to Start/Run and enter "MMC". Choose Add/Remove Snap-In. Click Add. Select 3 Snap-Ins: Component Services, Services, and SQLBase Services. Click Close, OK, and then save as MTS\_TD.

### **Setting Up COM+**

To add the Sample Bank package components and transactions

1. In the MMC open Component Services\Computer\My Computer\Com+ Applications.
2. Right click and select Add -> New Application.
3. Click 'Next' in the first page of the COM Application Installation Wizard.
4. Now select "Create an empty Application" button. Enter "TD Bank Demo" as the name of the application. And let this be a "Server application". Click Next.
5. Let it be "Interactive user" (in the Set Application Identity page). Click Finish.
6. Now, open "Bank Demo". Under this node, you will notice Components and Roles.
7. Click on Components. Do a right-click on Components. Select New -> Component. Click Next in the "COM Component Install Wizard". Select "Install new component".
8. Specify the full path for the DLL (that you have built for the component, 'MTS\_TD.DLL'. Click Next.

### **Executing the Bank client**

1. Make sure that Microsoft Distributed Transaction Coordinator (MS DTC) is running.
2. Make sure that SQLBase and SQLBrm are running.
3. Start the executable 'MTS\_TD\_Client\_with\_OLEDB.exe'
4. If you do have an 'Account' table in your ISLAND database then click the 'Remove Infrastructure' push button
5. Click the 'Create Infrastructure' pushbutton. This will create the 'Account' table and will enter two dummy accounts into the created table.

6. Click the 'New Account' pushbutton. This will create an account and the account number will be displayed in the appropriate field.
7. Click the 'Open Account' pushbutton. This will display the current account balance and enable/disable pushbuttons.
8. Enter an amount in the Deposit and Withdrawal Fields.
9. Click the 'Deposit' pushbutton. This should display that the deposit was done and the new balance should be displayed with appropriate messages.
10. Click the 'Withdrawal' pushbutton. This should display that the withdrawal was done and the new balance should be displayed with appropriate messages. Note: If sufficient funds are not available for the withdrawal an appropriate message will be displayed and no withdrawal will occur.
11. When you have finished with this account, click the 'Close Account Push button'.
12. Observe the DTC windows. You will notice that the component usage and transaction statistics windows have been updated.
13. Experiment with the bank client and observe the statistics using different transaction types, servers, and iterations. The first transaction takes longer than subsequent transaction for the following reasons:
  - a) The first transaction is creating the sample bank database tables and inserting temporary records.
  - b) Beginning the server process consumes system resources.
  - c) Opening database connections for the first time is a costly server operation.

To set the transaction attributes for your components

1. In the 'MTS\_TD.MTS\_QueryBalance' COM set the Transaction to 'Supported'
2. In the 'MTS\_TD.MTS\_TableDataOperations' COM set the transaction type to 'Required'
3. In the 'MTS\_TD.MTS\_TableOperations' COM set the transaction type to 'Requires New'
4. In the 'MTS\_TD.MTS\_UpdateBalance' COM set the transaction type to 'Required'

### [Download Source Code](#)

#### **Requirements**

- SQLBase 8.0
- SQLWindows 2.1