All About TransactionScope

S. M. Ahasan Habib
6 Jan 2014 CPOL

In this article, I explain with transaction related theory and code sample, how in various scenarios we can use TransactionScope with various options for managing real life transactions.

Download TransactionScope project - 12.7 KB

Table of Contents

1. Introduction

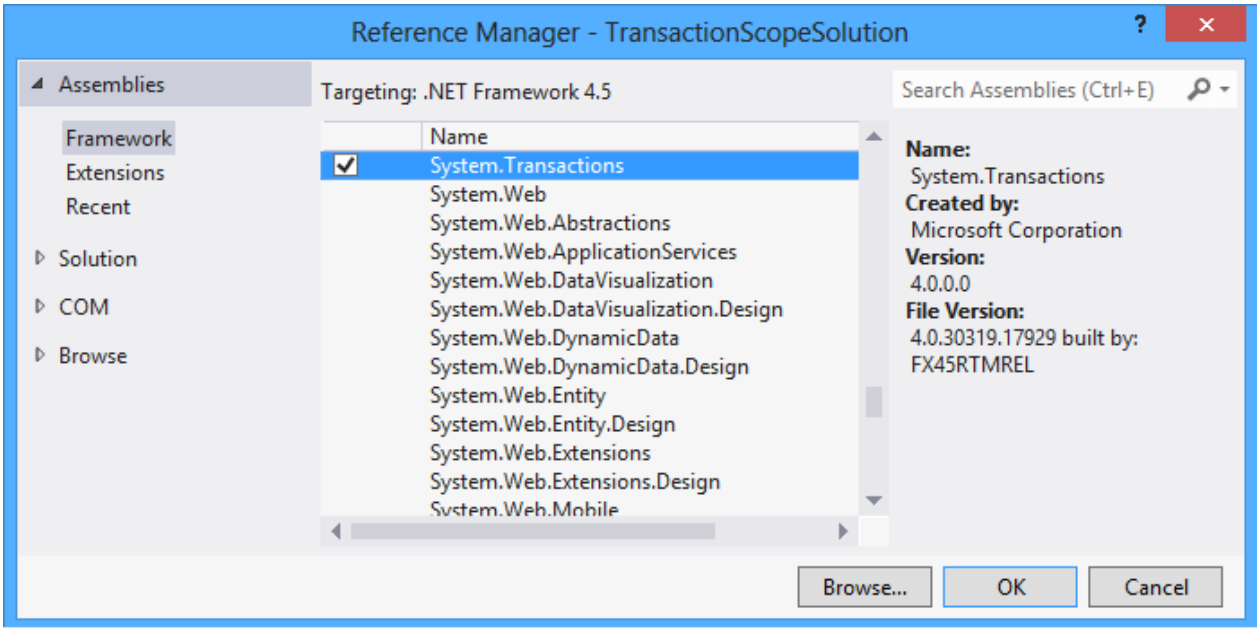
`TransactionScope` is a very special and important class in the .NET Framework. Supporting transactions from a code block is the main responsibility of this class. We often use this class for managing local as well as distributed transactions from our code. Use of `TransactionScope` is very simple and straightforward. It is very reliable and easy to use. For this reason it is very popular among .NET developers. In this article, I explain transaction related theory with code sample, and show various scenarios where we can use `TransactionScope` with various options for managing real life transactions.

2. Background

Transaction management is very, very important to any business application. Each and every large scale development framework provides a component for managing transactions. .NET Framework is a large development framework and it also provides its own transaction management component. Before the launch of .NET Framework 2.0 we used `SqlTransaction` to manage transactions. From version 2 .NET Framework has the `TransactionScope` class. This class is available in the System.Transactions assembly. This class provides a transactional framework with the help of which any .NET developer can write transactional code without knowing much details. For this reason it is very popular among .NET developers and they widely use it for managing transactions. But the story is not finished yet. I will say the story has only just started.

In the real world any one you will find exceptional scenarios, exceptional issues where only a knowledge of how to use `TransactionScope` is not good enough. To resolve transactional issues like deadlocks, timeouts, etc., you must know each and every concept directly/indirectly related to a transaction. There is no alternative. So the concepts of a transaction and its related components need to be clear.

3. How to Use TransactionScope




```
using(var scope = new System.Transactions.TransactionScope())
{
    //transactional code...
    //transactional code..
    scope.Complete();
}
```

Use of `TransactionScope` in a .NET application is very, very simple. Any one can use it by following these steps:

1. Add a System.Transactions assembly reference to the project.
2. Create a transactional scope/area with the help of the `TransactionScope` class starting with a `using` statement.
3. Writing code which needs to have transactional support.
4. Execute the `TransactionScope.Complete` method to commit and finish a transaction.

Really, as simple as that. But in a real life project only that knowledge is not sufficient. You need more transaction related knowledge, otherwise you can not handle transaction related issues. So first of all, we should be clear about the transactional concept.

4. Transaction

What is a transaction? You can find the definition of a transaction from various sources like Wikipedia, other websites, books, articles, blogs. In a very short, we can say, a series/number of works treated as a whole, either completed fully or not at all.

Example: Transfer money from Bank Account-A to Account-B

Series of (actually two) tasks/processes:

1. Withdraw amount from Account-A
2. Deposit that amount to Account-B

We understand that transfer of money (from Account-A to Account-B) consists of two individual processes. Transferring money will only be accurate and successful if both the processes are individually successful. If that is not happening, suppose process-1 succeeds but process-2 fails, then the money will be deducted from Account-A but not deposited to Account-B. If that happens, it will be very bad and no one will accept it.

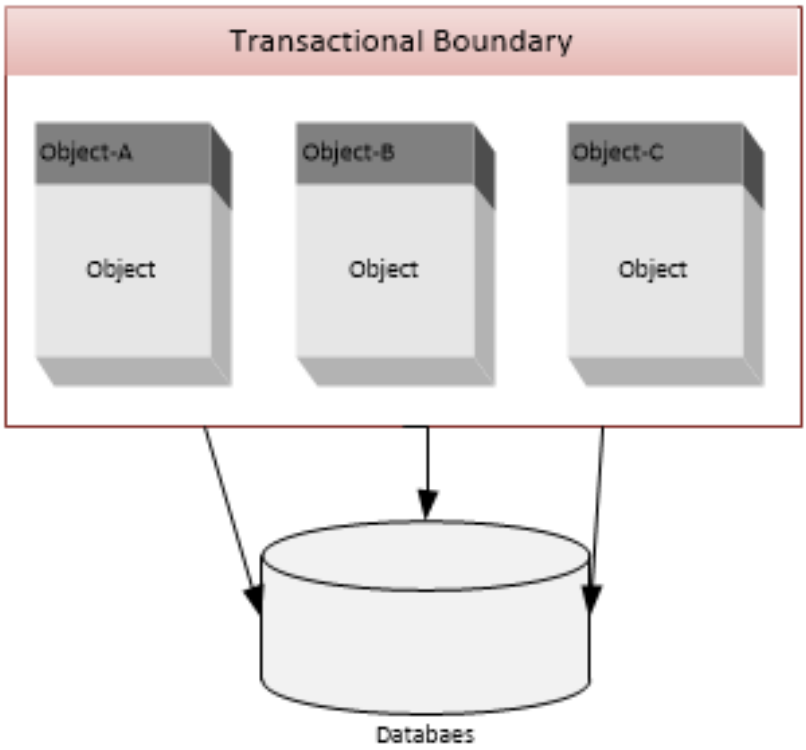
5. Business Transaction

Business transactions are interactions between Customer/Supplier/StackHolders and other parties who are involved in doing business. In this article I am not going to present anything regarding business transactions.

6. Database Transaction

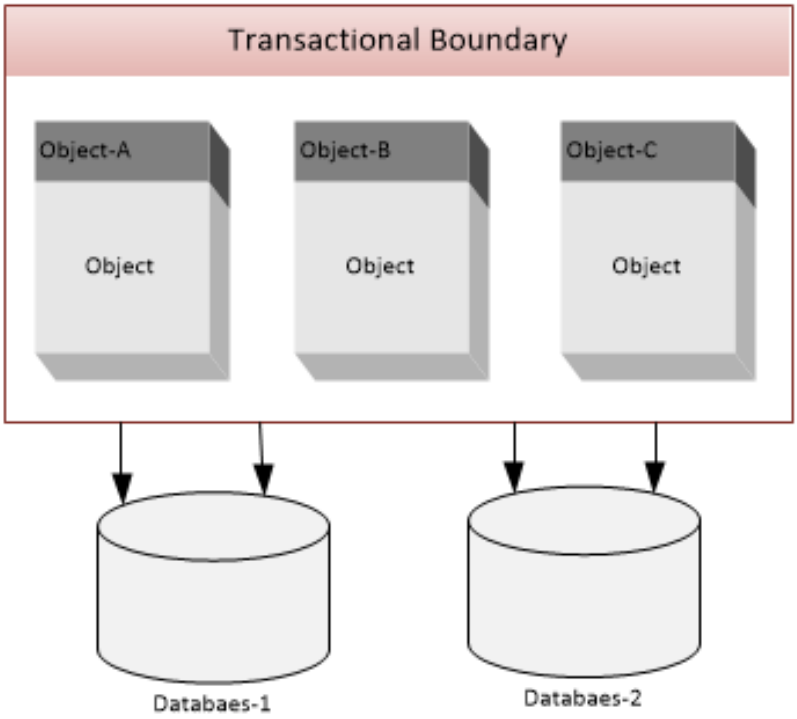
In software development, when we say transaction by default we guess that it is a database transaction. In a database transaction we can say, a series of data manipulation statements (insert/update/delete) execute as a whole. All statements either successfully execute, or will fail each and every statement, so that the database is in consistent mode. Database transactions actually represent a database state change in an accurate way.

7. Local Transaction



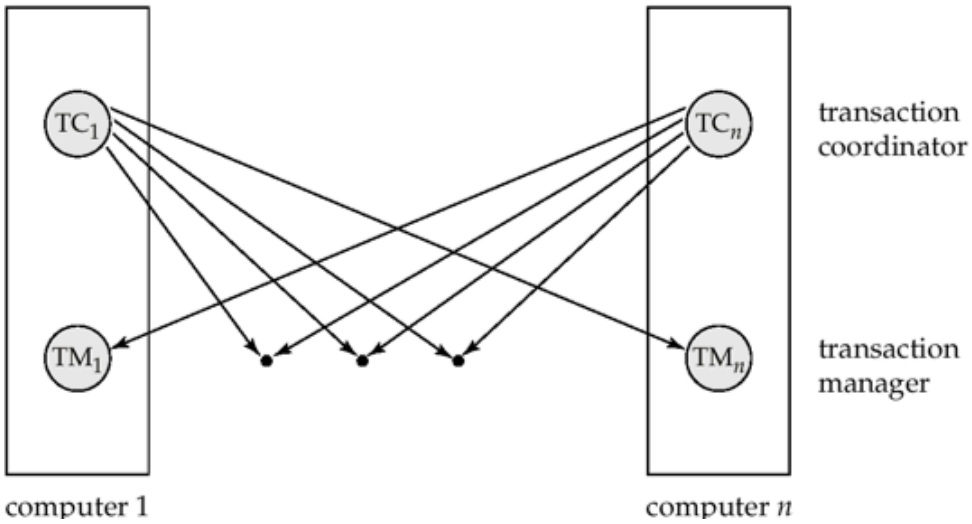
A transaction where a series of data manipulation statements execute as a whole on a single data source/database. It is actually a single phase transaction handled by a database directly. To manage local transactions, `System.Transactions` has a Lightweight Transaction Manager (LTM). It acts like a gateway. All transactions are started by `System.Transactions` are handled directly by this component. If it finds the transaction nature is distributed based on some predefined rules it has a fallback transaction to the MSDTC distributed transaction.

8. Distributed Transaction



A transaction which works with multiple data sources is called a distributed transaction. If a transaction fails then the affected data sources will be rolled back. In `System.Transactions`, MSDTC (Microsoft Distributed Transaction Coordinator) manages distributed transactions. It implements a two phase commit protocol. A distributed transaction is much slower than a local transaction. The transaction object automatically escalates a local transaction to a distributed transaction when it understands that a distributed transaction is needed. The developer can not do anything here.

9. Distributed Transaction System Architecture



We know that in a distributed transaction, several sites are involved. Each site has two components:

1. Transaction Manager
2. Transaction Coordinator

1. **Transaction Manager:** Maintains a log and uses that log if recovery is needed. It controls the whole transaction by initiating and completing, and managing the durability and atomicity of a transaction. It also coordinates transactions across one or more resources. There are two types of transaction managers.

1. **Local Transaction Manager:** Coordinates transaction over a single resource only.
2. **Gloabl Transaction Manager:** Coordinates transaction over multiple resources.

2. **Transaction Coordinator:** Starting the execution of transactions that originate at the site. Distributes subtransactions at appropriate sites so that they can execute in those sites. Coordinates each transaction of each site. As a result the transaction is committed or rolled back to all sites.

10. Connection Transaction

Transaction which is tied directly with a database connection (`SqlConnection`) is called Connection Transaction. `SqlTransaction` (`IDbTransaction`) is an example of a connection transaction. In .NET Framework 1.0/1.1 we use `SqlTransaction`.

C#

```
string connString = ConfigurationManager.ConnectionStrings["db"].ConnectionString;
using (var conn = new SqlConnection(connString))
{
    conn.Open();
    using (IDbTransaction tran = conn.BeginTransaction())
    {
        try
        {
            // transactional code...
            using (SqlCommand cmd = conn.CreateCommand())
            {
                cmd.CommandText = "INSERT INTO Data(Code) VALUES('A-100');";
                cmd.Transaction = tran as SqlTransaction;
                cmd.ExecuteNonQuery();
            }
            tran.Commit();
        }
        catch(Exception ex)
        {
            tran.Rollback();
            throw;
        }
    }
}
```

11. Ambient Transaction

A transaction which automatically identifies a code block that needs to support a transaction without explicitly mentioning any transaction related things. An ambient transaction is not tied just to a database, any transaction aware provider can be used. `TransactionScope` implements an ambient transaction. If you see the use of TransactionScope, you will not find transaction related anything sent to any method or setting any property. A code block is automatically attached with the transaction if that code is in any TransactionScope. A WCF transaction is another example of a transaction aware provider. Any one can write a transaction aware provider like the WCF implementation.

12. Transaction Properties

There are four important properties for a transaction. We call them ACID properties. They are:

1. **A**-Atomic
2. **C**-Consistent
3. **I**-Isolation
4. **D**-Durable

1. **Atomic**: If all parts of the transaction individually succeed then data will be committed and the database will be changed. If any single part of a transaction fails then all parts of the transaction will fail and the database will remain unchanged. Part of the transaction might fail for various reasons like business rule violation, power failure, system crash, hardware failure, etc.

2. **Consistent**: Transaction will change the database from one valid state to another valid state following various database rules like various data integrity constraints (primary/unique key, check/not null constraint, referential integrity with valid reference, cascading rules) etc.
3. **Isolation**: One transaction will be hidden from another transaction. In another way we can say, one a transaction will not affect an other transaction if both work concurrently.
4. **Durability**: After a transaction is successfully completed (committed to the database), changed data will not be lost in any situation like system failure, database crash, hardware failure, power failure etc.

13. Transaction Isolation Level

Now I will start explaining a very important thing directly related to transactions, and that is transaction isolation level. Why is it so important? First of all, I previously explained that isolation is an important transaction property. It describes each transaction is isolated from another and do not affect other concurrently executed transactions. How does a transaction management system achieve that important feature?

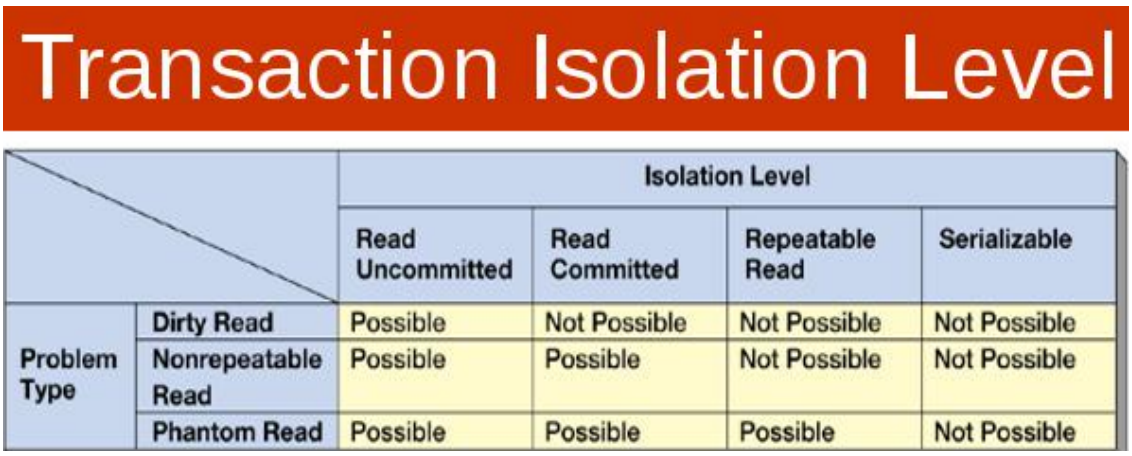
A Transaction Management System introduces a locking mechanism. With the help of this mechanism one transaction is isolated from another. The locking policy behaves differently based on the Isolation level set for each transaction. There are four very important isolation levels in .NET transaction scope. These are:

1. Serializable
2. Repeatable Read
3. Read Committed
4. Read UnComitted

Before I start explaining isolation levels, I need to explain data reading mechanizm inside a transaction. This data reading mechanism is very important to understand isolation levels properly.

- **Dirty Read:** One transaction reads changed data of anohter tranaction but that data is still not committed. You may take decission/action based on that data. A problem will arise when data is rolled-back later. If rollback happens then your decision/action will be wrong and it produces a bug in your application.
- **Non Repeatable Read:** A transaction reads the same data from same table multiple times. A problem will arise when for each read, data is different.
- **Phantom Read:** Suppose a transaction will read a table first and it finds 100 rows. A problem will arise when the same tranaction goes for another read and it finds 101 rows. The extra row is called a phantom row.

Now I will start explaining in short the important isolation levels:



1. **Serializable:** Highest level of isolation. It locks data exclusively when read and write occurs. It acquires range locks so that phantom rows are not created.
2. **Repeatable Read:** Second highest level of isolation. Same as serializable except it does not acquire range locks so phantom rows may be created.
3. **Read Committed:** It allow shared locks and read only committed data. That means never read changed data that are in the middle of any transaction.
4. **Read Un-Committed:** It is the lowest level of Isolation. It allows dirty read.

Now I will start explaining `TransactionScope` *and its usage pattern:*

14. TranactionScope Default Properties

It is very important to know about the default properties of the `TransactionScope` object. Why? Because many times we create and use this object without configuring anything.

Three very important properties are:

1. `IsolationLevel`
2. `Timeout`
3. `TransactionScopeOptions`

We create and use `TransactionScope` as follows:

C#

```
using (var scope = new TransactionScope())
{
    //transctional code…
    scope.Complete();
}
```

Here the `TransactionScope` object is created with the default constructor. We did not define any value for `IsolationLevel`, `Timeout`, and `TransactionScopeOptions`. So it gets default values for all three properties. So now we need to know what the default property values of these properties.

Property	Default Value	Available Options
`IsolationLevel`	Serializable	Serializable, Read Committed, Read Un Committed, Repeatable Read
`Timeout`	1 Minute	Maximum 10 Minutes
`TransactionScopeOption`	Required	Required, Required New, Suppress

1. **Isolation Level:** It defines the locking mechanism and policy to read data inside another transaction.
2. **Timeout:** How much time object will wait for a transaction to be completed. Never confuse it with the `SqlCommand Timeout` property. `SqlCommand Timeout` defines how much time the `SqlCommand` object will wait for a database operation (select/insert/update/delete) to be completed.
3. **TransactionScopeOption:** It is an enumeration. There are three options available in this enumeration:

No	Option	Description
1	`Required`	It is default value for `TransactionScope`. If any already exists any transaction then it will join with that transaciton otherwise create new one.
2	`RequiredNew`	When select this option a new transaction is always created. This transaction is independent with its outer transaction.
3	`Suppress`	When select this option, no transaction will be created. Even if it already

How to know the default values of these properties?

The System.Transactions assembly has two classes:

1. **Transaction**
2. **TransactionManager**

These classes will provide default values. Inside `TransactionScope`, if you run the following code, you will know the default values:

C#

```
using (var scope = new System.Transactions.TransactionScope())
{
```

```
    IsolationLevel isolationLevel = Transaction.Current.IsolationLevel;
    TimeSpan defaultTimeout = TransactionManager.DefaultTimeout;
    TimeSpan maximumTimeout = TransactionManager.MaximumTimeout;
}
```

Is it possible to override the default property values?

Yes, you can. Suppose you want the default value to be 30 seconds and the maximum timeout value to be 20 minutes. If that is the requirement then you can do it using your web config.

XML

```
<system.transactions>
    <defaultSettings timeout="30"/>
    <machineSettings maxTimeout="1200"/>
</system.transactions>
```

For the `machineSettings` value, you need to update your *machine.config* in your server.

HTML

```
<section name="machineSettings" type="System.Transactions.Configuration.MachineSettingsSection,
System.Transactions,Version=2.0.0.0,Culture=neutral,PublicKeyToken=b77a5c561934e089,
Custom=null"allowdefinition="MachineOnly"allowexedefinition="MachineToApplication" />
```

15. Transaction Isolation Level Selection

You need to have a proper knowledge when you use isolation levels. The following table will give you a very basic idea so that you can understand the basics and select the appropriate isolation level for your transaction scope.

Isolation Level	Suggestion
Serializable	It locks data exclusively at the time of read/write operations. For that reason, many times it may create a deadlock, and as a result you may get a timeout exception. You can use this isolation level for a highly secured transactional application like a financial application.
Repeatable Read	Same as Serializable except allows phantom rows. May use in a financial application or a heavily transactional application but need to know where phantom row creational scenarios are not there.
Read Committed	Most of the applications you can use it. SQL Server default isolation level is this.
Read Un-Committed	Applications with these have no need to support concurrent transactions.

Now I will explain with scenarios, how we can use TransactionScope:

16. Requirement-1

Create a transaction in which isolation level will be read committed and transaction timeout will be 5 minutes.

Implementation:

C#

```
var option = new TransactionOptions();
option.IsolationLevel = IsolationLevel.ReadCommitted;
option.Timeout = TimeSpan.FromMinutes(5);
using (var scope = new TransactionScope(TransactionScopeOption.Required, option))
{
    ExcuteSQL("CREATE TABLE MyNewTable(Id int);");
    scope.Complete();
}
```

First off, create `TransactionOptions` and set `ReadCommitted` and 5 minutes to its `IsolationLevel` and `Timeout` property, respectively.

Second, create a transactional block by creating a `TransactionScope` object with its parameterized constructor. In this constructor you will pass a `TransactionOptions` object which you created early and the `TransactionScopeOption.Required` value.

One important note, many times we are confused when using a DDL statement (Data Definition Language) in a transaction and a question arises, will it support DDL transaction? The answer is yes. You can use a DDL statement like create/alter/ drop statement in the transaction. You can even use a **`Truncate`** statement inside the transaction.

17. Requirement-2

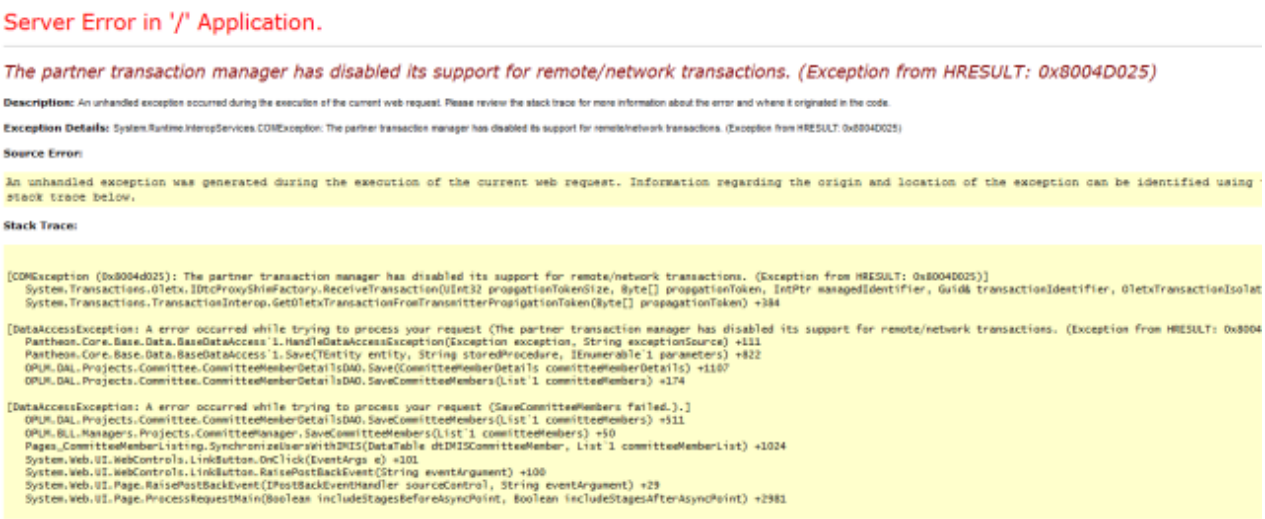
We need to create a transaction where a database operation will be in my local database and another will be in a remote database.

Implementation:

```
using (var scope = new TransactionScope())
{
    UpdateLocalDatabase();
    UpdateRemoteDatabase();
    scope.Complete();
}
```

There is no difference between a local or remote/distributed transaction implementation code in transactions. Previously I said that `TransactionScope` implements ambient type transaction. This means, it automatically marks code blocks that need to support a transaction, local or remote. But you may find an error when working with distributed transactions. The error message will be like:

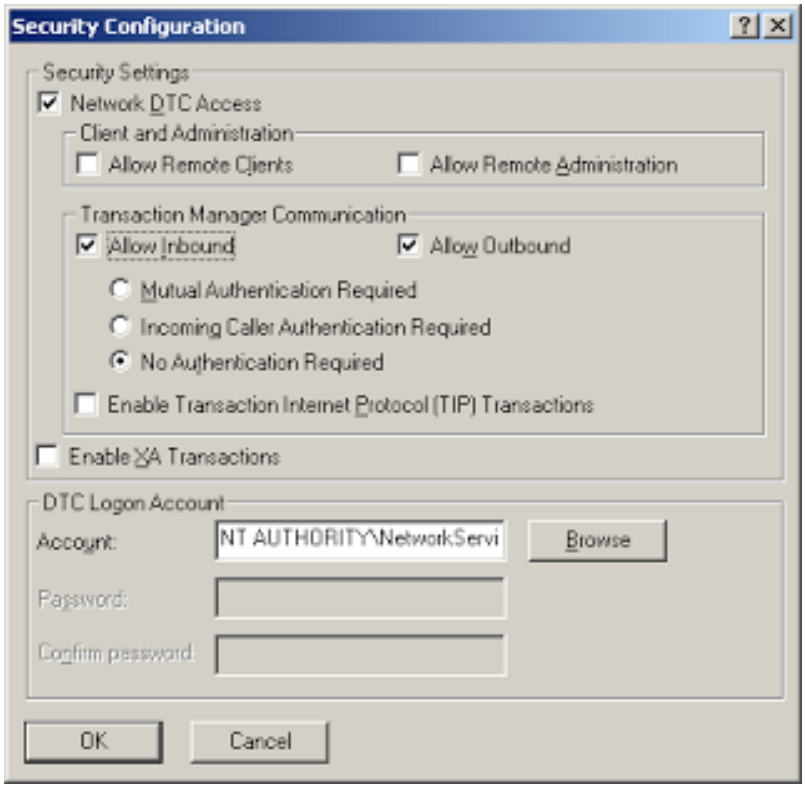
"*The partner transaction manager has disabled its support for remote/network transaction.*"



If you find that type of exception, you need to configure security settings, both your local and remote servers, for MSDTC, and make sure services are running.

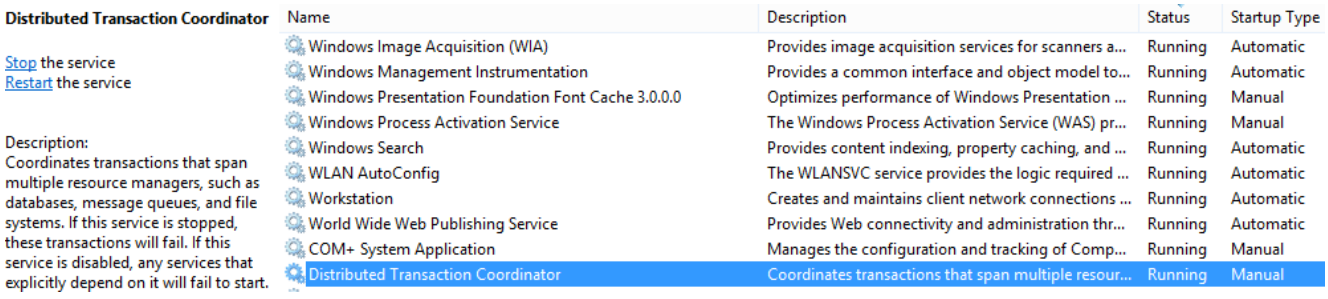
To find the MSDTC configuration interface, you will go to:

ControlPanel* > *AdministritiveTools* >*ComponentServices* > *DistributedTransactionCoordinator* > *LocalDTC



Some options for the Security tab are described bellow:

Property Name	Description
Network DTC Access	If not selected, MSDTC will not allow any remote transaction
Allow Remote Clients	If it is checked, MSDTC will allow to coordinate remote clients for transaction.
Allow Remote Administration	Allow remote computers to access and configure these settings.
Allow Inbound	Allow computers to flow transaction to local computers. This option is needed where MSDTC is hosted for a resource manager like SQL Server.
Allow Outbound	Allow computers to flow transaction to remote computers. It is needed for a client computer where transaction is initiated.
Mutual Authentication	Local and Remote computers communicate with encrypted messages. They establish a secured connection with the Windows Domain Account for message communication.
Incoming Calling Authentication Required	If mutual authentication cannot be established but the incoming caller is authenticated then communication will be allowed. It supports only Windows 2003/XP ServicePack-2.
No Authentication Required	It allows any non-authenticated non-encrypted communication.
Enable XA Transaction	Allows different operating systems to communicate with MSDTC with XA Starndard.
DTC Logon Account	DTC Service running account. Default account is Network Service.



18. Distributed Transaction Performance

Distributed transactions are slower than local transactions. A two phase commit protocol is used for managing distributed transactions. A two phase commit protocol is nothing but an algorithm by which a distributed transaction is performed. There are three commit protocols that are mostly used:

1. **Auto Commit:** Transaction is committed automatically if all SQL statements are executed successfully or rolled-back if any of them fails to execute.
2. **Two Phase Commit:** Transaction waits before final commit for messages from all other parties involved in transaction. It locks resources before commit or rollback. For this reason it is called a blocking protocol. In terms of performance it is the reason it is much slower. It is a widely used protocol for managing distributed transactions.
3. **Three Phase Commit:** Transaction is finally committed if all nodes are agreed. It is a non-blocking protocol. In terms of performance it is faster than the two phase commit protocol. This protocol is complicated and more expensive but avoids some drawbacks in the two phase commit protocol.

19. Requirement-3

I want to create a transaction inside another transaction.

Implementation:

C#

```
string connectionString = ConfigurationManager.ConnectionStrings["db"].ConnectionString;
var option = new TransactionOptions
{
     IsolationLevel = IsolationLevel.ReadCommitted,
     Timeout = TimeSpan.FromSeconds(60)
};
using (var scopeOuter = new TransactionScope(TransactionScopeOption.Required, option))
{
    using (var conn = new SqlConnection(connectionString))
    {
        using (SqlCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText="INSERT INTO Data(Code, FirstName)VALUES('A-100','Mr.A')";
            cmd.Connection.Open();
            cmd.ExecuteNonQuery();
        }
    }
    using (var scopeInner = new TransactionScope(TransactionScopeOption.Required, option))
    {
        using (var conn = new SqlConnection(connectionString))
        {
            using (SqlCommand cmd = conn.CreateCommand())
            {
                cmd.CommandText="INSERT INTO Data(Code, FirstName) VALUES('B-100','Mr.B')";
                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
            }
        }
        scopeInner.Complete();
    }
    scopeOuter.Complete();
}
```

No problems in creating a transaction inside anohter (nested) transaction. You should define the behaviour or the inner transaction. This behaviour is dependent on the value of `TransactionScopeOption`. If you select `Required` as `TransactionScopeOption`, it will join its outer transaction. That means if the outer transaction is committed then the inner transaction will commit if the outer transaction is rolled back, then the inner transcation will be rolled back. If you select the `RequiredNew` value of `TrasnactionScopeOption`, a new transaction will be created and this transaction will

independently be committed or rolled back. You must be clear about those concepts before working with nested transactions using `TransactionScope`.

20. Requirement-4

I want to call rollback explicitly from a transaction.

Implementation:

C#

```
using (var scope = new TransactionScope())
{
    //either 1 of following lines will use
    Transaction.Current.Rollback();
    scope.Dispose();
    //if you comment the following line transaction will
    //automatically be rolled back
    //scope.Complete();
}
```

If you do not call the `TransactionScope.Complete()` method then the transaction will automatically be rolled back. If you need to explicitly call rollback for some scenarios, then you have two options:

1. Executing `Transaction.Current.Rollback()` will rollback the current transaction.
2. Executing `TransactionScope.Dispose()` will also rollback the current transaction.

Just one thing: remember that if you explicitly call `Transaction.Rollback()` or `TranactionScope.Dispose()` then you should not call the `TransactionScope.Complete()` method. If you do so then you will get an `ObjectDisposeException`.

"Cannot access a disposed object. Object name 'TransactionScope'"

21. Requirement-5

I want to create a file/folder dynamically inside a transaction scope. If my transaction is rolled back then I want that created file/folder to be removed automatically, like a database row.

Implementation:

C#

```
string newDirectory = @"D:\TestDirectory";
string connectionString = ConfigurationManager.ConnectionStrings["db"].ConnectionString;
using (var scope = new TransactionScope())
{
    using (var conn = new SqlConnection(connectionString))
    {
        using (SqlCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText = "Insert into data(Code) values ('A001');";
            cmd.Connection.Open();
            cmd.ExecuteNonQuery();
        }
    }
    Directory.CreateDirectory(newDirectory);
    File.Create(@"D:\NewFile.txt").Dispose();
    scope.Dispose();
}
```

`TranactionScope` is not limited for only databases. It will support other data sources like FileSystem, MSMQ, etc. But you need more work to support TranactionScope. First of all what I show in the above code block **will not work**. Why? Because that directory

creation and file creation will not be marked for transaction by default. Then what do we need to do?

C#

```
public interface IEnlistmentNotification
{
    void Commit(Enlistment enlistment);
    void InDoubt(Enlistment enlistment);
    void Prepare(PreparingEnlistment preparingEnlistment);
    void Rollback(Enlistment enlistment);
}
```

The `System.Transactions` namespace has an interface named `IEnlistmentNotification`. If I want my component/service to be transaction aware then I need to implement that interface. The following code will show a very simple and straightforward way to implement this:

C#

```
public class DirectoryCreator : IEnlistmentNotification
{
    public string _directoryName;
    private bool _isCommitSucceed = false;
    public DirectoryCreator(string directoryName)
    {
        _directoryName = directoryName;
        Transaction.Current.EnlistVolatile(this, EnlistmentOptions.None);
    }
    public void Commit(Enlistment enlistment)
    {
        Directory.CreateDirectory(_directoryName);
        _isCommitSucceed = true;
        enlistment.Done();
    }
    public void InDoubt(Enlistment enlistment)
    {
        enlistment.Done();
    }
    public void Prepare(PreparingEnlistment preparingEnlistment)
    {
        preparingEnlistment.Prepared();
    }
    public void Rollback(Enlistment enlistment)
    {
        if (_isCommitSucceed))
            Directory.Delete(_directoryName);
        enlistment.Done();
    }
}
```

The above class will create a directory (folder) and this component is transaction aware. We can use this class with any TranactionScope and if TranactionScope is committed the directory will be created, otherwise it will be deleted (if already created). I show here just the diretory creation, if you want you can create a class/component for file creation. Now, how to use this class in the transactions scope?

C#

```
string newDirectory = @"D:\TestDirectory";
string connectionString = ConfigurationManager.ConnectionStrings["db"].ConnectionString;
using (var scope = new TransactionScope())
{
    using (var conn = new SqlConnection(connectionString))
    {
        using (SqlCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText = "Insert into data(Code) values ('A001');";
            cmd.Connection.Open();
            cmd.ExecuteNonQuery();
```

```
        }
    }
    var creator = new DirectoryCreator(newDirectory);
    Transaction.Current.Rollback();
    //scope.Complete();
}
```

Now, it will work!

Transactional NTFS(TxF) .NET is a open source project. You can use this library for creating/writing/coping file/directory inside transactionscope and it will support transaction automatically.

- You first need to download component from http://txfnet.codeplex.com
- Add that component as reference to your project.
- Use component api to your transactional block.

Txf API usage code sample:

C#

```
using (var ts = new System.Transactions.TransactionScope())
{
    using (var conn = new SqlConnection(connectionString))
    {
        using (SqlCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText = "Insert into data(Code) values ('A001');";
            cmd.Connection.Open();
            cmd.ExecuteNonQuery();
        }
    }
    TxF.Directory.CreateDirectory("d:\\xyz", true);
    TxF.File.CreateFile("D:\\abc.txt", File.CreationDisposition.OpensFileOrCreate);
    ts.Complete();
}
```

TxF component supports:

- Create/Delete Directory
- Create/Delete File
- Read/Write File
- Copy File

22. Points of Interest

Transaction Management is actually a huge subject. It is a very complex subject too, specially distributed transaction. I tried my level best to present it as simple as possible. If you want to get all transaction related knowledge then you should study more on that. I suggest you read research papers on transactions, specially distributed transactions.

You can also explore more regarding transaction aware service/components. I showed here a very simple way to implement them. But in real life you may face difficult scenarios. So you need to prepare for that. In the near future Microsoft may add transaction aware components like dictionary/filesystem/directory service, etc. If it that happens then developers' life will be more easier.

Sample Source Code

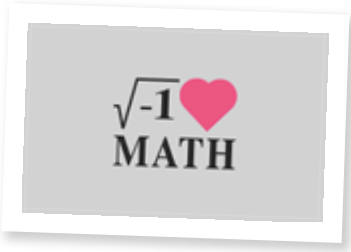
I have attached a source code sample with this article. I wrote this source code with the help of Visual Studio 2012 with .NET Framework 4.5. I added a Unit Test project so that you can debug/test the code and properly understand it.

References

- http://msdn.microsoft.com/en-us/library/ms172152(v=vs.110).aspx
- http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope(v=vs.90).aspx
- http://en.wikipedia.org/wiki/ACID
- http://stackoverflow.com/questions/974596/what-is-a-transaction
- http://msdn.microsoft.com/en-us/library/system.transactions.transactionscopeoption(v=vs.110).aspx
- http://stackoverflow.com/questions/224689/transactions-in-net
- http://technet.microsoft.com/en-us/library/dd145385.aspx
- http://msdn.microsoft.com/en-us/magazine/cc163388.aspx

License

About the Author



S. M. Ahasan Habib

Architect
Bangladesh

How do I describe myself to you? How can I explain that this is true?
I am who I am because of you! My work I love you !!

Comments and Discussions

68 messages have been posted for this article Visit **https://www.codeproject.com/Articles/690136/All-About-TransactionScope** to post and view comments on this article, or click **here** to get a print view with messages.