This page is specific to

Microsoft Visual Studio 2010/.NET Framework 4

.NET Framework Class Library

**TransactionScope Class**

Makes a code block transactional. This class cannot be inherited.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifInheritance Hierarchy

[System..::.Object](http://msdn.microsoft.com/en-us/library/system.object.aspx)  
**System.Transactions..::.TransactionScope**

**Namespace:** [System.Transactions](http://msdn.microsoft.com/en-us/library/system.transactions.aspx)  
**Assembly:** System.Transactions (in System.Transactions.dll)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifSyntax

Visual Basic

Public NotInheritable Class TransactionScope \_

Implements IDisposable

C#

public sealed class TransactionScope : IDisposable

Visual C++

public ref class TransactionScope sealed : IDisposable

F#

[<Sealed>]

type TransactionScope =

class

interface IDisposable

end

The **TransactionScope** type exposes the following members.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifConstructors

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Description: Public method | [TransactionScope()()()](http://msdn.microsoft.com/en-us/library/1hfx49zc.aspx) | Initializes a new instance of the **TransactionScope** class. |
| Description: Public method | [TransactionScope(Transaction)](http://msdn.microsoft.com/en-us/library/ms149851.aspx) | Initializes a new instance of the **TransactionScope** class and sets the specified transaction as the ambient transaction, so that transactional work done inside the scope uses this transaction. |
| Description: Public method | [TransactionScope(TransactionScopeOption)](http://msdn.microsoft.com/en-us/library/k6ksw5zc.aspx) | Initializes a new instance of the **TransactionScope** class with the specified requirements. |
| Description: Public method | [TransactionScope(Transaction, TimeSpan)](http://msdn.microsoft.com/en-us/library/ms149852.aspx) | Initializes a new instance of the **TransactionScope** class with the specified timeout value, and sets the specified transaction as the ambient transaction, so that transactional work done inside the scope uses this transaction. |
| Description: Public method | [TransactionScope(TransactionScopeOption, TimeSpan)](http://msdn.microsoft.com/en-us/library/9wykw3s2.aspx) | Initializes a new instance of the **TransactionScope** class with the specified timeout value and requirements. |
| Description: Public method | [TransactionScope(TransactionScopeOption, TransactionOptions)](http://msdn.microsoft.com/en-us/library/ms149853.aspx) | Initializes a new instance of the **TransactionScope** class with the specified requirements. |
| Description: Public method | [TransactionScope(Transaction, TimeSpan, EnterpriseServicesInteropOption)](http://msdn.microsoft.com/en-us/library/ms149854.aspx) | Initializes a new instance of the **TransactionScope** class with the specified timeout value and COM+ interoperability requirements, and sets the specified transaction as the ambient transaction, so that transactional work done inside the scope uses this transaction. |
| Description: Public method | [TransactionScope(TransactionScopeOption, TransactionOptions, EnterpriseServicesInteropOption)](http://msdn.microsoft.com/en-us/library/ms149855.aspx) | Initializes a new instance of the **TransactionScope** class with the specified scope and COM+ interoperability requirements, and transaction options. |

[Top](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.aspx#mainBody)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifMethods

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Description: Public method | [Complete](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.complete.aspx) | Indicates that all operations within the scope are completed successfully. |
| Description: Public method | [Dispose](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.dispose.aspx) | Ends the transaction scope. |
| Description: Public method | [Equals(Object)](http://msdn.microsoft.com/en-us/library/bsc2ak47.aspx) | Determines whether the specified [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx) is equal to the current [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx). (Inherited from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx).) |
| Description: Protected method | [Finalize](http://msdn.microsoft.com/en-us/library/system.object.finalize.aspx) | Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx).) |
| Description: Public method | [GetHashCode](http://msdn.microsoft.com/en-us/library/system.object.gethashcode.aspx) | Serves as a hash function for a particular type. (Inherited from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx).) |
| Description: Public method | [GetType](http://msdn.microsoft.com/en-us/library/system.object.gettype.aspx) | Gets the [Type](http://msdn.microsoft.com/en-us/library/system.type.aspx) of the current instance. (Inherited from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx).) |
| Description: Protected method | [MemberwiseClone](http://msdn.microsoft.com/en-us/library/system.object.memberwiseclone.aspx) | Creates a shallow copy of the current [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx). (Inherited from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx).) |
| Description: Public method | [ToString](http://msdn.microsoft.com/en-us/library/system.object.tostring.aspx) | Returns a string that represents the current object. (Inherited from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx).) |

[Top](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.aspx#mainBody)

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifRemarks

The [System.Transactions](http://msdn.microsoft.com/en-us/library/system.transactions.aspx) infrastructure provides both an explicit programming model based on the [Transaction](http://msdn.microsoft.com/en-us/library/system.transactions.transaction.aspx) class, as well as an implicit programming model using the **TransactionScope** class, in which transactions are automatically managed by the infrastructure.

|  |
| --- |
| **Description: Important noteImportant** |
| It is recommended that you create implicit transactions using the **TransactionScope** class, so that the ambient transaction context is automatically managed for you. You should also use the **TransactionScope** and [DependentTransaction](http://msdn.microsoft.com/en-us/library/system.transactions.dependenttransaction.aspx) class for applications that require the use of the same transaction across multiple function calls or multiple thread calls. For more information on this model, see the [Implementing An Implicit Transaction Using Transaction Scope](http://msdn.microsoft.com/en-us/library/ms172152.aspx) topic. For more information on writing a transactional application, see [Writing A Transactional Application](http://msdn.microsoft.com/en-us/library/ms229973.aspx). |

Upon instantiating a **TransactionScope** by the **new** statement, the transaction manager determines which transaction to participate in. Once determined, the scope always participates in that transaction. The decision is based on two factors: whether an ambient transaction is present and the value of the **TransactionScopeOption** parameter in the constructor. The ambient transaction is the transaction your code executes in. You can obtain a reference to the ambient transaction by calling the static [Current](http://msdn.microsoft.com/en-us/library/system.transactions.transaction.current.aspx) property of the [Transaction](http://msdn.microsoft.com/en-us/library/system.transactions.transaction.aspx) class. For more information on how this parameter is used, please see the "Transaction Flow Management" section of the [Implementing An Implicit Transaction Using Transaction Scope](http://msdn.microsoft.com/en-us/library/ms172152.aspx) topic.

If no exception occurs within the transaction scope (that is, between the initialization of the **TransactionScope** object and the calling of its [Dispose](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.dispose.aspx) method), then the transaction in which the scope participates is allowed to proceed. If an exception does occur within the transaction scope, the transaction in which it participates will be rolled back.

When your application completes all work it wants to perform in a transaction, you should call the [Complete](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.complete.aspx) method only once to inform that transaction manager that it is acceptable to commit the transaction. Failing to call this method aborts the transaction.

A call to the [Dispose](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.dispose.aspx) method marks the end of the transaction scope. Exceptions that occur after calling this method may not affect the transaction.

If you modify the value of [Current](http://msdn.microsoft.com/en-us/library/system.transactions.transaction.current.aspx) inside a scope, an exception is thrown when [Dispose](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.dispose.aspx) is called. However, at the end of the scope, the previous value is restored. In addition, if you call [Dispose](http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.dispose.aspx) on [Current](http://msdn.microsoft.com/en-us/library/system.transactions.transaction.current.aspx) inside a transaction scope that created the transaction, the transaction aborts at the end of the scope.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExamples

The following example demonstrates how to use the **TransactionScope** class to define a block of code to participate in a transaction.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl90_ctl00_ctl00_code');" \o "Copy Code)

' This function takes arguments for 2 connection strings and commands to create a transaction

' involving two SQL Servers. It returns a value > 0 if the transaction is committed, 0 if the

' transaction is rolled back. To test this code, you can connect to two different databases

' on the same server by altering the connection string, or to another 3rd party RDBMS

' by altering the code in the connection2 code block.

Public Function CreateTransactionScope( \_

ByVal connectString1 As String, ByVal connectString2 As String, \_

ByVal commandText1 As String, ByVal commandText2 As String) As Integer

' Initialize the return value to zero and create a StringWriter to display results.

Dim returnValue As Integer = 0

Dim writer As System.IO.StringWriter = New System.IO.StringWriter

Try

' Create the TransactionScope to execute the commands, guaranteeing

' that both commands can commit or roll back as a single unit of work.

Using scope As New TransactionScope()

Using connection1 As New SqlConnection(connectString1)

' Opening the connection automatically enlists it in the

' TransactionScope as a lightweight transaction.

connection1.Open()

' Create the SqlCommand object and execute the first command.

Dim command1 As SqlCommand = New SqlCommand(commandText1, connection1)

returnValue = command1.ExecuteNonQuery()

writer.WriteLine("Rows to be affected by command1: {0}", returnValue)

' If you get here, this means that command1 succeeded. By nesting

' the using block for connection2 inside that of connection1, you

' conserve server and network resources as connection2 is opened

' only when there is a chance that the transaction can commit.

Using connection2 As New SqlConnection(connectString2)

' The transaction is escalated to a full distributed

' transaction when connection2 is opened.

connection2.Open()

' Execute the second command in the second database.

returnValue = 0

Dim command2 As SqlCommand = New SqlCommand(commandText2, connection2)

returnValue = command2.ExecuteNonQuery()

writer.WriteLine("Rows to be affected by command2: {0}", returnValue)

End Using

End Using

' The Complete method commits the transaction. If an exception has been thrown,

' Complete is called and the transaction is rolled back.

scope.Complete()

End Using

Catch ex As TransactionAbortedException

writer.WriteLine("TransactionAbortedException Message: {0}", ex.Message)

Catch ex As ApplicationException

writer.WriteLine("ApplicationException Message: {0}", ex.Message)

End Try

' Display messages.

Console.WriteLine(writer.ToString())

Return returnValue

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl90_ctl00_ctl01_code');" \o "Copy Code)

// This function takes arguments for 2 connection strings and commands to create a transaction

// involving two SQL Servers. It returns a value > 0 if the transaction is committed, 0 if the

// transaction is rolled back. To test this code, you can connect to two different databases

// on the same server by altering the connection string, or to another 3rd party RDBMS by

// altering the code in the connection2 code block.

static public int CreateTransactionScope(

string connectString1, string connectString2,

string commandText1, string commandText2)

{

// Initialize the return value to zero and create a StringWriter to display results.

int returnValue = 0;

System.IO.StringWriter writer = new System.IO.StringWriter();

try

{

// Create the TransactionScope to execute the commands, guaranteeing

// that both commands can commit or roll back as a single unit of work.

using (TransactionScope scope = new TransactionScope())

{

using (SqlConnection connection1 = new SqlConnection(connectString1))

{

// Opening the connection automatically enlists it in the

// TransactionScope as a lightweight transaction.

connection1.Open();

// Create the SqlCommand object and execute the first command.

SqlCommand command1 = new SqlCommand(commandText1, connection1);

returnValue = command1.ExecuteNonQuery();

writer.WriteLine("Rows to be affected by command1: {0}", returnValue);

// If you get here, this means that command1 succeeded. By nesting

// the using block for connection2 inside that of connection1, you

// conserve server and network resources as connection2 is opened

// only when there is a chance that the transaction can commit.

using (SqlConnection connection2 = new SqlConnection(connectString2))

{

// The transaction is escalated to a full distributed

// transaction when connection2 is opened.

connection2.Open();

// Execute the second command in the second database.

returnValue = 0;

SqlCommand command2 = new SqlCommand(commandText2, connection2);

returnValue = command2.ExecuteNonQuery();

writer.WriteLine("Rows to be affected by command2: {0}", returnValue);

}

}

// The Complete method commits the transaction. If an exception has been thrown,

// Complete is not called and the transaction is rolled back.

scope.Complete();

}

}

catch (TransactionAbortedException ex)

{

writer.WriteLine("TransactionAbortedException Message: {0}", ex.Message);

}

catch (ApplicationException ex)

{

writer.WriteLine("ApplicationException Message: {0}", ex.Message);

}

// Display messages.

Console.WriteLine(writer.ToString());

return returnValue;

}