Using Async/Await Task Methods With SQL Queries .NET 4.5

JamesAmos, 2 Sep 2016

Microsoft .NET 4.5 introduced new "async and await" methods to provide an easy way of implementing asynchronisity using .NET tasks objects.

Download demo - 24.9 KB

Introduction

Microsoft .NET 4.5 introduced new "`async` and `await`" methods to provide an easy way of implementing asynchronisity using .NET "Task" objects.

This allows developers to make `async` calls more flexibly, as opposed to standard threading/callback methods.

In this article, I've built a demo to show how this can be applied to build easy to use SQL functions which can be called `async`.

The full demo and source code are available from this article.

Github

The code is also available on GitHub. You are welcome to create issues and pull requests.

Getting Started

For this article, we shall be using the .NET SQL Client library for connecting to a Microsoft SQL Server database, however the same methodology of wrapping up code in a task can be used with any other type of connection.

In this example, we will wrap our code which connects to the database to get data into a function which returns a "Task" object based on the data type we require.

Building Asynchronous Task Functions

We start by building the initial wrapper which returns a task object. We shall be returning a dataset in this example.

Side note: Any data type can be returned with task objects, as such you could return a `string`, integer or a custom class.

The following is the basic wrapper for our function. This bit of code will return a task object based on the datatype, with a section to add our own code to build and return the dataset.

```
public Task<DataSet> GetDataSetAsync(string sConnectionString, string sSQL)
{
    return Task.Run(() =>
    {
        //YOUR CODE HERE
    });
}
```

We can now insert code which connects to the server to get data based on a query into this function.

Example 1: Function to Fill a dataset Based on the SQL Query

In this example, code has been added to the body of the function which uses a standard SQL connection `string` and objects from the `System.Data.SQLClient` .NET library to connect to the database and fill a dataset based on the query.

Once the dataset has been filled, the code will then return the `dataset` using a standard "`return`" call.

```
public Task<DataSet> GetDataSetAsync
(string sConnectionString, string sSQL, params SqlParameter[] parameters)
{
    return Task.Run(() =>
    {
        using (var newConnection = new SqlConnection(sConnectionString))
        using (var mySQLAdapter = new SqlDataAdapter(sSQL, newConnection))
        {
            mySQLAdapter.SelectCommand.CommandType = CommandType.Text;
            if (parameters != null) mySQLAdapter.SelectCommand.Parameters.AddRange(parameters);

            DataSet myDataSet = new DataSet();
            mySQLAdapter.Fill(myDataSet);
            return myDataSet;
        }
    });
}
```

We can now call this function asynchronously to return a dataset of data based on our query.

Example 2: Function to Execute a Command, Passing Back the Rows Affected

In this example, we have changed the data type to integer, as we simply want to pass back the rows affected for the result of this function.

```
public Task<int> ExecuteAsync(string sConnectionString, string sSQL, params SqlParameter[]
parameters)
{
    return Task.Run(() =>
    {
        using (var newConnection = new SqlConnection(sConnectionString))
        using (var newCommand = new SqlCommand(sSQL, newConnection))
        {
            newCommand.CommandType = CommandType.Text;
            if (parameters != null) newCommand.Parameters.AddRange(parameters);

            newConnection.Open();
            return newCommand.ExecuteNonQuery();
        }
    });
}
```

Further to this, there are existing async task functions on the SqlCommand object which can be used to simplify the function further.

```
public async Task<int> ExecuteAsync(string sConnectionString,
string sSQL, params SqlParameter[] parameters)
{
    using (var newConnection = new SqlConnection(sConnectionString))
    using (var newCommand = new SqlCommand(sSQL, newConnection))
    {
        newCommand.CommandType = CommandType.Text;
        if (parameters != null) newCommand.Parameters.AddRange(parameters);

        await newConnection.OpenAsync().ConfigureAwait(false);
        return await newCommand.ExecuteNonQueryAsync().ConfigureAwait(false);
    }
}
```

We can use this function to execute SQL and get the rows affected.

Calling Asynchronous Task Functions

Now that we have an `async` function to get data from SQL, we can now call this asynchronously using the `async\await` clause from any method.

To call the function to get data, we start by adding "`async`" into the method declaration.

```
private async <type> MyMethod()
```

Now that our method is an "`async`" method, we can use the "`await`" option on the tasks returned from our SQL functions.

Execute Task Function

To execute task functions, you need to add "`await`" before the function call.

This will execute the function, and return the "`Result`" to the variable should one be specified.

```
private async Task GetSomeData(string sSQL)
{
    //Use Async method to get data
    DataSet results = await GetDataSetAsync(sConnectionString, sSQL, sqlParams);

    //Populate once data received
    grdResults.DataSource = results.Tables[0];
}
```

Variables do not need to be specified. You can call the function using "`await`" which will run the task without retrieving a result.

```
private async Task ExecuteSomeData(string sSQL)
{
    //Use Async method to get data
    await ExecuteAsync(sConnectionString, sSQL, sqlParams);
}
```

The "`await`" option can be specified multiple times if you need to make other asyncronous calls.

You can make GUI updates before and after the "`await`" command, as it is only the `await` command which is run asynchronously.

The simplest use of "`async`/`await`" is to use the "`await`" option, then continue with your code afterwards.

However, the .NET task object provides other methods of continuation once the asynchronous task has completed.

Running Multiple Tasks/Chaining Tasks

If you need to run multiple SQL operations, there are a few ways of achieving this.

Specifying Tasks As Variables Then Running Together

You can specify a number of commands as variables and run them all at once asynchronously, then get the results as necessary.

This can be achieved by setting up your tasks as variables.

Once setup, you will need to add them into an array then use the "`Task.WhenAll`" to run all the tasks asynchronously.

You can then access the results on the task object after completion.

```
//Setup tasks
Task<int> ExecuteTask1 = database.ExecuteAsync(sConnectionString, sExecuteSQL1, sqlParams);
Task<int> ExecuteTask2 = database.ExecuteAsync(sConnectionString, sExecuteSQL2, sqlParams);
Task<int> ExecuteTask3 = database.ExecuteAsync(sConnectionString, sExecuteSQL3, sqlParams);
//Add to array
Task<int>[] Tasks = new Task<int>[] { ExecuteTask1, ExecuteTask2, ExecuteTask3 };
//Run all
await Task.WhenAll(Tasks);
//Get results
int iRowsAffected = Tasks[0].Result + Tasks[1].Result + Tasks[2].Result;
```

Calling Other Async Tasks With "ContinueWith"

Another method of chaining the tasks together is using the "`ContinueWith`" method on the task object returned by your SQL function.

In this example, we start by running an execute task function, then once complete, we run a select task function to get a dataset to a variable.

```
DataSet results = await database.ExecuteAsync(sConnectionString, sExecuteSQL,
sqlParams).ContinueWith
(t => database.GetDataSetAsync(sConnectionString, sGetSQL, sqlParams)).Result;
```

Calling Standard Methods With "ContinueWith"

You can call a standard method on the "`ContinueWith`" method.

In this example, we use "`ContinueWith`" on the task object to pass the data through to a separate method.

```
private async Task RunSQLQuery(string sSQL)
{
    //Use await method to get data
    await GetDataSetAsync
         (sConnectionString, sSQL, sqlParams).ContinueWith(t => PopulateResults(t.Result));
}
private void PopulateResults(DataSet results)
{
    //Do something with results
}
```

WARNING: When using "`ContinueWith`" in this way, this will continue the `async` thread, meaning that if you intend to update GUI components such as a `grid` or `textbox` control, you may encounter the error:

*Cross-thread operation not valid: Control **MyControl** accessed from a thread other than the thread it was created on.*

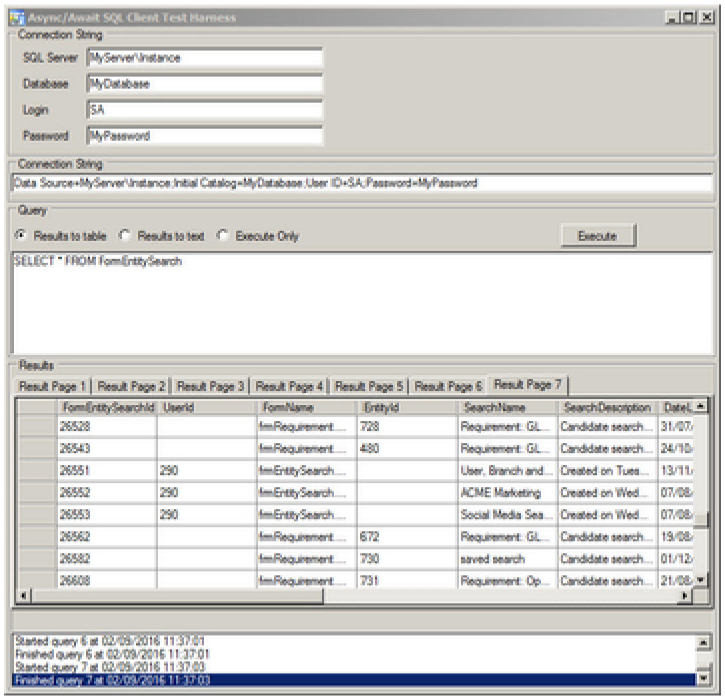
In cases like this, you can invoke the method from within the "`ContinueWith`" which will allow for GUI updates.

```
await database.GetDataSetAsync(sConnectionString, sSQL, sqlParams).ContinueWith
(t => this.Invoke((Action)(() => { PopulateResultsToScreen(t.Result); })));
```

Demo Project

A simple demo project has been added to this article to demonstrate the `async`/`await` options. The application will require you to enter server details to connect to an existing database.

You can enter your SQL query, then use the "Execute" button to fire the SQL query.



Connection String

SQL Server: MyServer\Instance
Database: MyDatabase
Login: SA
Password: MyPassword

Connection String

Data Source=MyServer\Instance;Initial Catalog=MyDatabase;User ID=SA;Password=MyPassword

Query

[✓ Results to table] Results to text Execute Only

```
SELECT * FROM FormEntitySearch
```

Results

Result Page 1 | Result Page 2 | Result Page 3 | Result Page 4 | Result Page 5 | Result Page 6 | Result Page 7

FormEntitySearchId	UserId	FormName	EntityId	SearchName	SearchDescription	DateL
26528		frmRequirement...	728	Requirement: GL...	Candidate search...	31/07/
26543		frmRequirement...	480	Requirement: GL...	Candidate search...	24/10/
26551	290	frmEntitySearch...		User, Branch and...	Created on Tues...	13/11/
26552	290	frmEntitySearch...		ACME Marketing	Created on Wed...	07/08/
26553	290	frmEntitySearch...		Social Media Sea...	Created on Wed...	07/08/
26562		frmRequirement...	672	Requirement: GL...	Candidate search...	19/08/
26582		frmRequirement...	730	saved search	Candidate search...	01/12/
26608		frmRequirement...	731	Requirement: Op...	Candidate search...	21/08/

```
Started query 6 at 02/09/2016 11:37:01
Finished query 6 at 02/09/2016 11:37:01
Started query 7 at 02/09/2016 11:37:03
Finished query 7 at 02/09/2016 11:37:03
```

The code is stored in the "`frmTestForm`" form, which starts from the "`Execute`" method if you wanted to see this code in action.

License

About the Author

JamesAmos

Software Developer TriSys Business Software

United Kingdom

No Biography provided

Comments and Discussions

5 messages have been posted for this article Visit **https://www.codeproject.com/Articles/1121822/Using-Async-Await-Task-Methods-With-SQL-Queries-NE** to post and view comments on this article, or click **here** to get a print view with messages.