

Creating the Middle Layer Part 2

In the last session we created the link between the presentation layer and the middle layer by creating our first class. The class contains a public function that will contain the code for the Delete method. We also have a parameter on the function allowing us to copy data from the presentation layer to the middle layer.

In this session we will create the link between the middle layer and the data layer completing the delete function of the system.

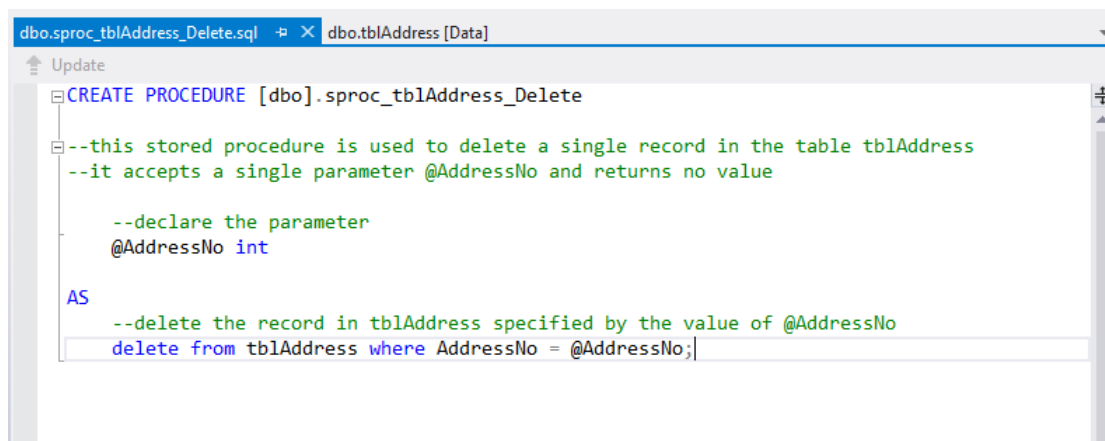
A Quick Re-Cap

Before we do anything else we need to remind ourselves what is going on in the data layer.

Inside the data layer we have a table called tblAddress containing some test data.

	AddressNo	HouseNo	Street	Town	PostCode	CountyCode	DateAdded	Active
▶	1	1		Some Street	LE1 1BE	34	07/08/2013	True
	2	22		The Road	N19 6FF	48	07/08/2013	True
	3	33	The Mews	High Street	LE1 6FG	34	07/08/2013	True
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

We have also created a stored procedure called sproc_tblAddress_Delete.

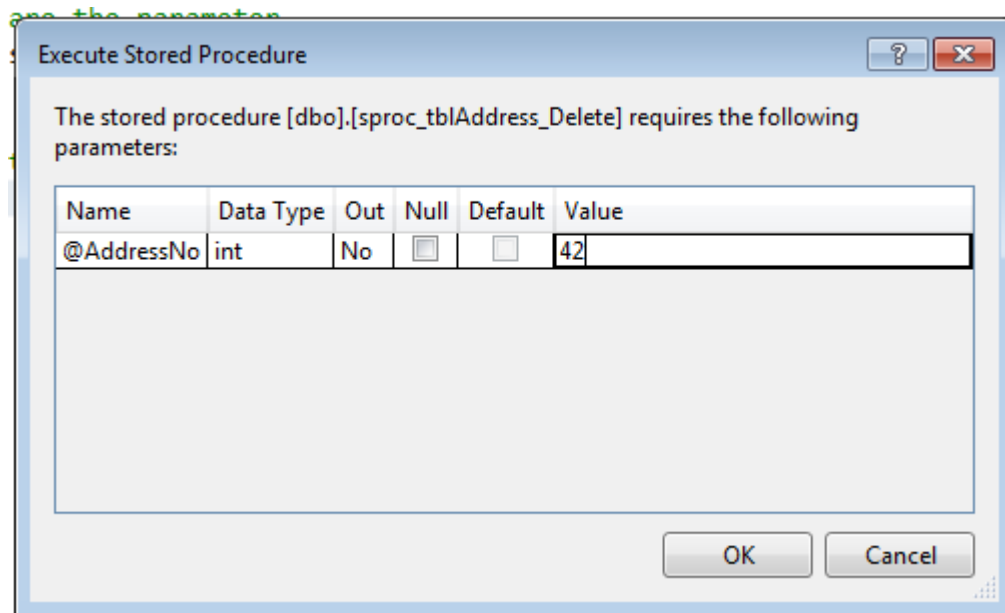
The screenshot shows a SQL Server Enterprise Manager window with two tabs: 'dbo.sproc_tblAddress_Delete.sql' and 'dbo.tblAddress [Data]'. The 'dbo.sproc_tblAddress_Delete.sql' tab is active, displaying the following T-SQL code:

```
CREATE PROCEDURE [dbo].sproc_tblAddress_Delete
--this stored procedure is used to delete a single record in the table tblAddress
--it accepts a single parameter @AddressNo and returns no value
--declare the parameter
@AddressNo int
AS
--delete the record in tblAddress specified by the value of @AddressNo
delete from tblAddress where AddressNo = @AddressNo;
```

This stored procedure accepts a single parameter @AddressNo which is used in the SQL query to delete the record.

```
delete from tblAddress where AddressNo = @AddressNo;
```

If we execute the query in the data layer it will ask for a value to use in the parameter and delete that record...

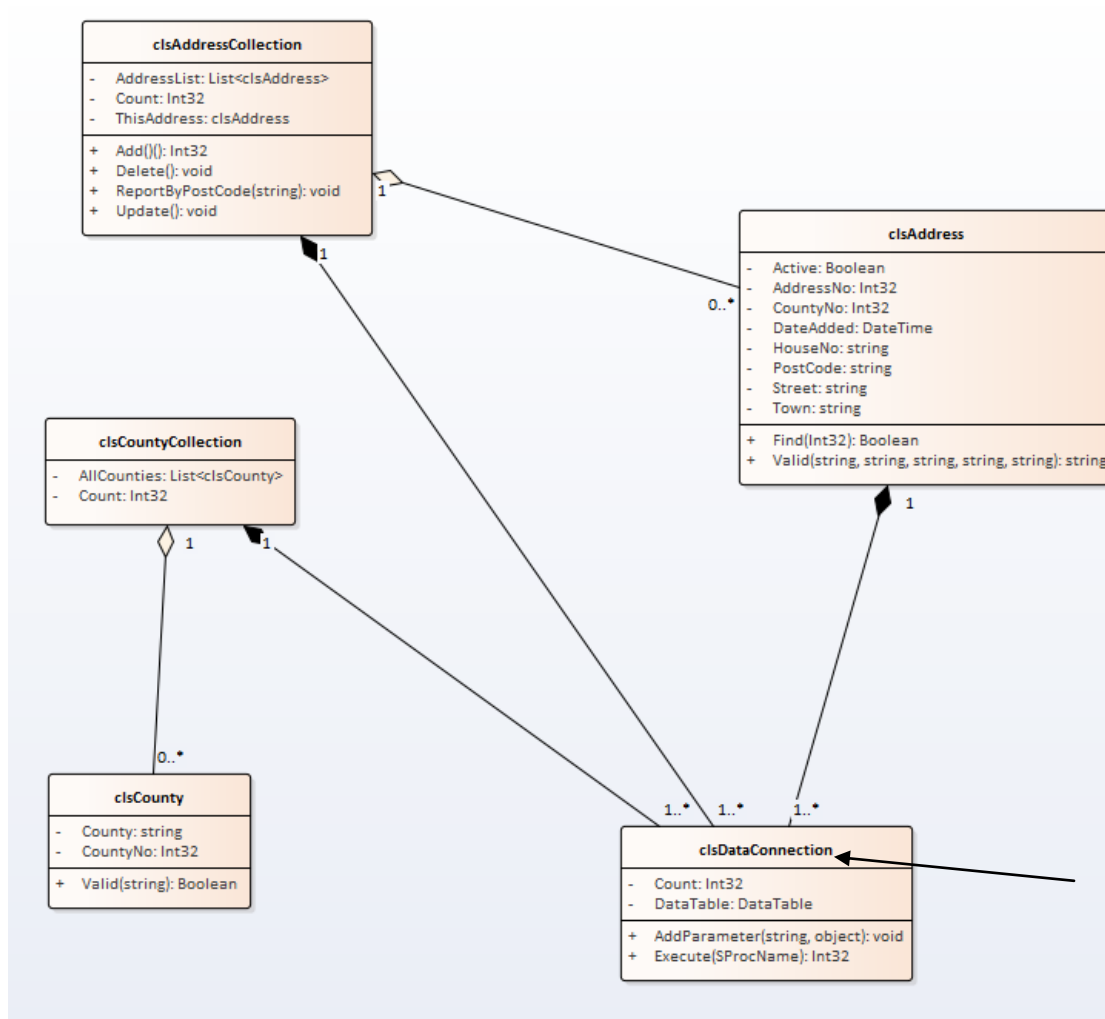


The question is “how do we link the middle layer code to our data layer?”

The answer is that we need a second class file that has been written to allow our middle layer talk to the data layer.

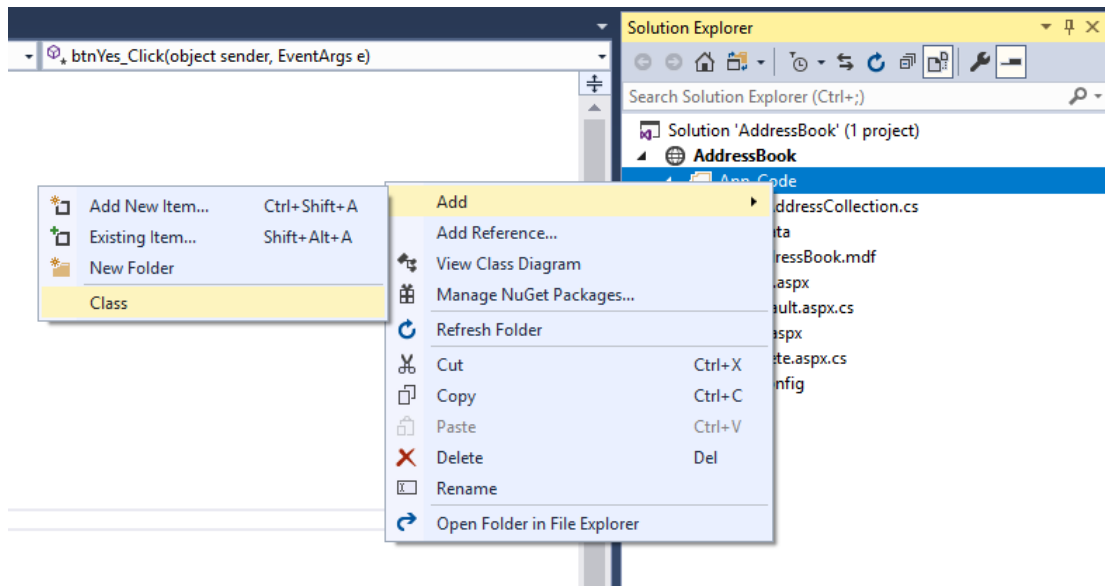
Creating the Data Connection Class File

The data connection class fits into the class diagram like so...

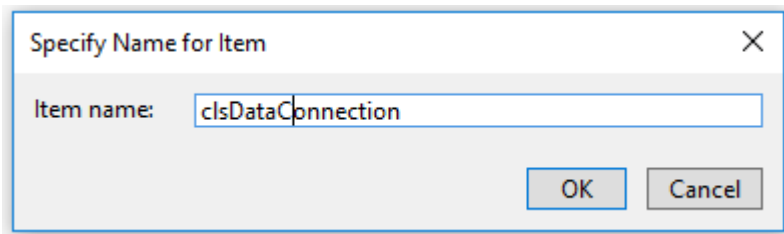


The data connection class provides all of the connectivity to the database for any class that needs it.

To create the data connection class file in right click on the App_Code folder and select Add – Class.



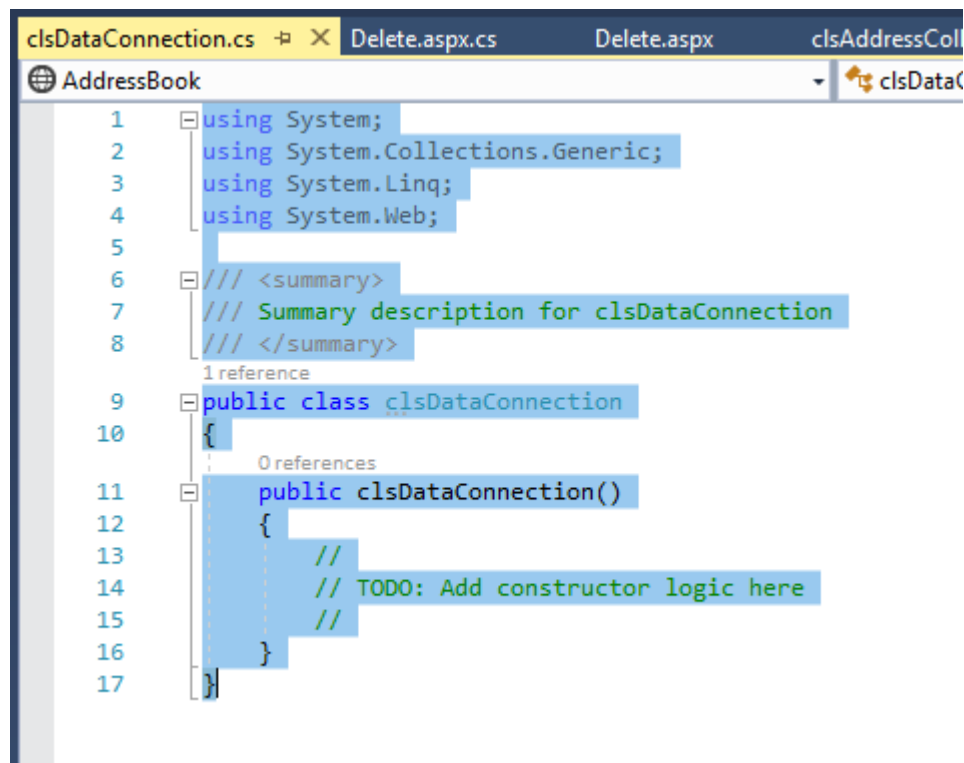
For the name of the class enter clsDataConnection



Press OK.

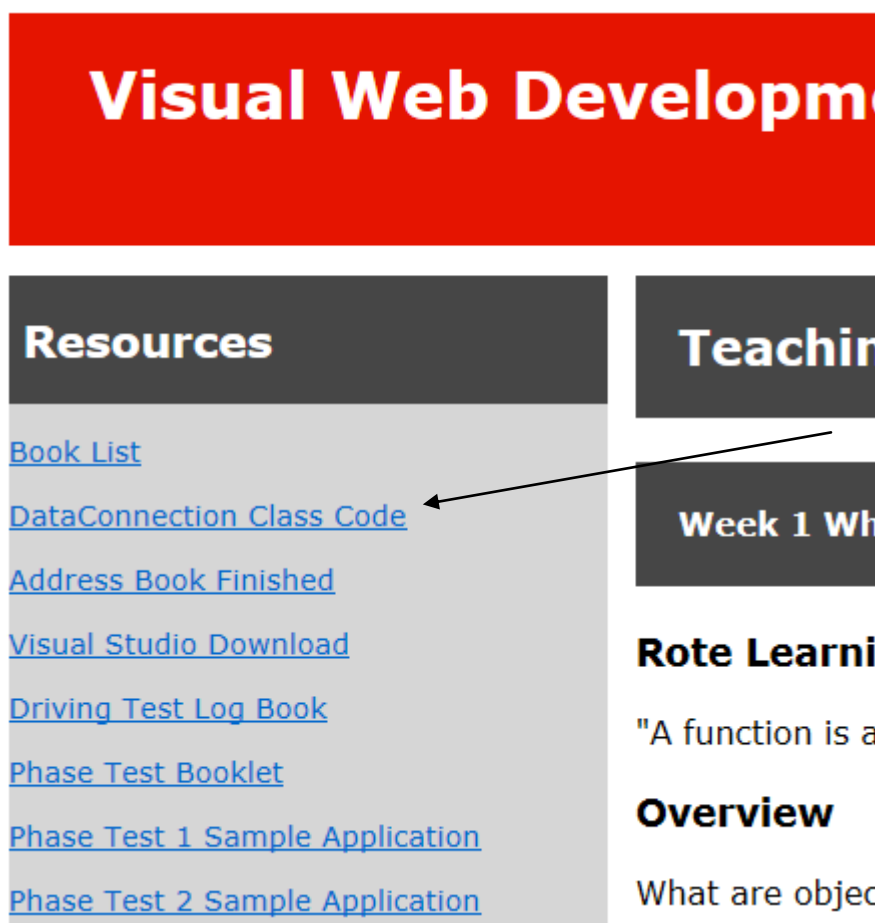
As with the first class it will create some code automatically in this case we want to replace it all.

Highlight the code in the class file created for you and delete it all.



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5
6 /// <summary>
7 /// Summary description for clsDataConnection
8 /// </summary>
9 public class clsDataConnection
10 {
11     public clsDataConnection()
12     {
13         //
14         // TODO: Add constructor logic here
15         //
16     }
17 }
```

Next obtain the code for the class file from the module web site...



Visual Web Development

Resources

- [Book List](#)
- [DataConnection Class Code](#)
- [Address Book Finished](#)
- [Visual Studio Download](#)
- [Driving Test Log Book](#)
- [Phase Test Booklet](#)
- [Phase Test 1 Sample Application](#)
- [Phase Test 2 Sample Application](#)

Teaching

Week 1 Wh

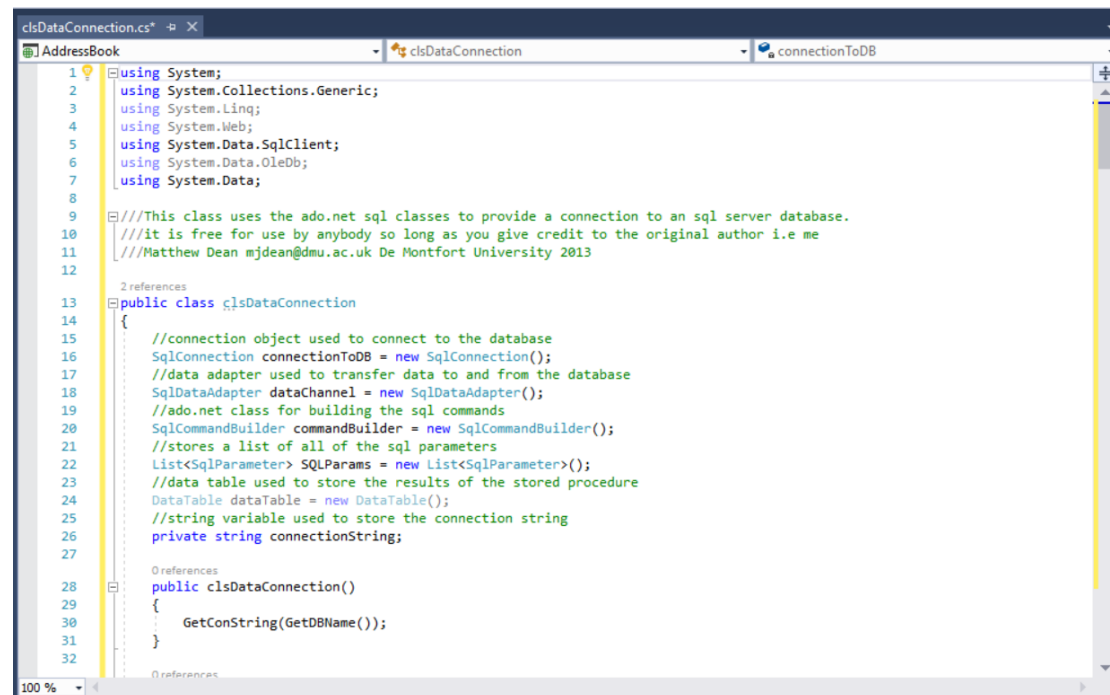
Rote Learning

"A function is a

Overview

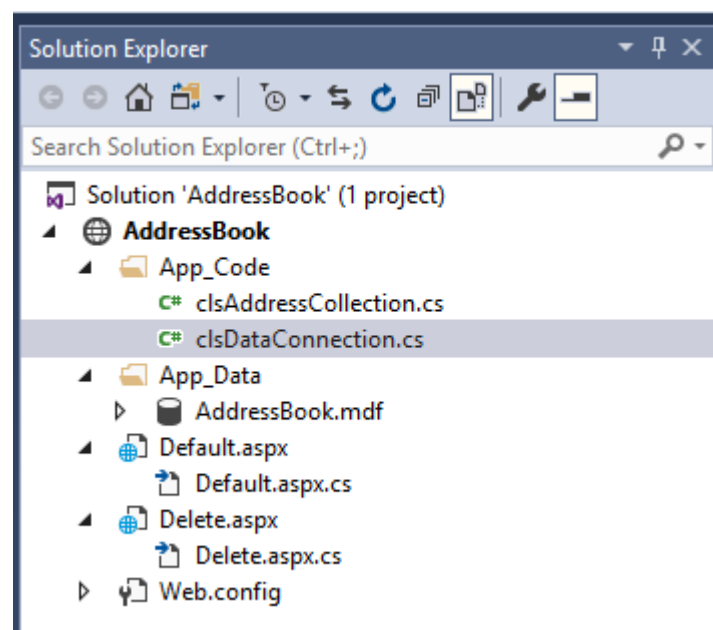
What are object

Paste it all into this blank class file...



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Data.SqlClient;
6 using System.Data.OleDb;
7 using System.Data;
8
9
10 ///This class uses the ado.net sql classes to provide a connection to an sql server database.
11 ///it is free for use by anybody so long as you give credit to the original author i.e me
12 ///Matthew Dean mjdean@dmu.ac.uk De Montfort University 2013
13
14 2 references
15 public class clsDataConnection
16 {
17     //connection object used to connect to the database
18     SqlConnection connectionToDB = new SqlConnection();
19     //data adapter used to transfer data to and from the database
20     SqlDataAdapter dataChannel = new SqlDataAdapter();
21     //ado.net class for building the sql commands
22     SqlCommandBuilder commandBuilder = new SqlCommandBuilder();
23     //stores a list of all of the sql parameters
24     List<SqlParameter> SQLParams = new List<SqlParameter>();
25     //data table used to store the results of the stored procedure
26     DataTable dataTable = new DataTable();
27     //string variable used to store the connection string
28     private string connectionString;
29
30     References
31     public clsDataConnection()
32     {
33         GetConnectionString(GetDBName());
34     }
35
36     0 references
```

Close and save the class definition. It should be listed in the solution explorer.



Making the Link from Middle Layer to Data Layer

This class has been specifically written for the module to make connecting to the database as simple as possible. The code is smart and will work out the name of your database by looking at the contents of the App_Data folder.

In this example we are going to use two of the class' methods.

AddParameter This method allows us to add a parameter for use in a stored procedure.

Execute This method tells the object to execute a specific stored procedure.

We are going to make use of this class code within the address book class we created last week.

Creating and Using our Object

Open the class we created last week and modify the code for the delete function like so...

```
public Boolean Delete(Int32 AddressNo)
{
    ///this public function provides the functionality for the delete method

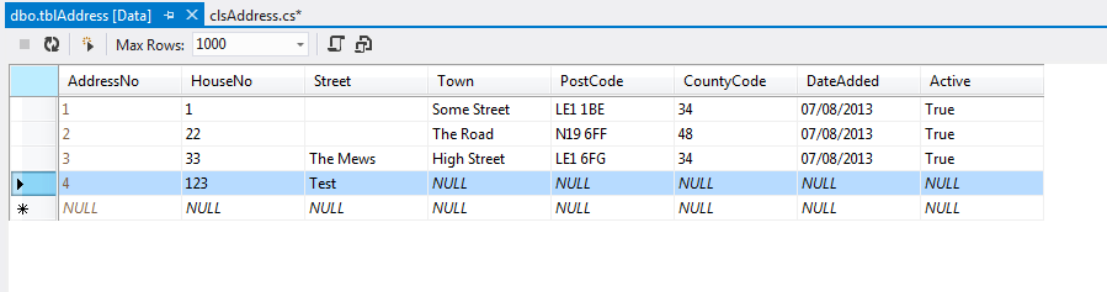
    ///create an instance of the data connection class called MyDatabase
    clsDataConnection MyDatabase = new clsDataConnection();
    ///add the AddressNo parameter passed to this function to the list of parameters to use in the database
    MyDatabase.AddParameter("@AddressNo", AddressNo);
    ///execute the stored procedure in the database
    MyDatabase.Execute("sproc_tblAddress_Delete");
    ///set the return value for the function
    return true;
}
```

In this code we are doing the following

1. We create an instance of clsDataConnection called MyDatabase
2. We use the AddParameter method to add the data contained in AddressNo to the parameter in the stored procedure called @AddressNO
3. We invoke the Execute method to run the stored procedure

We shall test this with the debugger to see if everything works ok.

Before running the program make sure that you have some data in your table.



	AddressNo	HouseNo	Street	Town	PostCode	CountyCode	DateAdded	Active
1	1			Some Street	LE1 1BE	34	07/08/2013	True
2	22		The Road		NI9 6FF	48	07/08/2013	True
3	33		The Mews	High Street	LE1 6FG	34	07/08/2013	True
4	123		Test	NULL	NULL	NULL	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Make a note of the primary key value for the record you wish to delete (in this example 4).

Next place a breakpoint in the presentation layer so that we may pause code execution.

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    //create an instance of the class clsAddressBook called MyAddressBook
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //declare a variable to store the address number to delete
    Int32 AddressNo;
    //copy the data from the interface to the RAM converting the data to Int32
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //invoke the delete method of the object passing it the data entered by the user
    MyAddressBook.Delete(AddressNo);
}
```

Run the program and enter the primary key value of the record you want to delete.

Are you sure you want to delete this address?

Press the Yes button to trigger the click event handler for the button and the break point should pause execution.

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    //create an instance of the class clsAddressBook called MyAddressBook
    clsAddressBook MyAddressBook = new clsAddressBook();
    //declare a variable to store the address number to delete
    Int32 AddressNo;
    //copy the data from the interface to the RAM converting the data to Int32
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //invoke the delete method of the object passing it the data entered by the user
    MyAddressBook.Delete(AddressNo);
}
```

Press F10 until you get to the call to the Delete method.

```
AddressNo = Convert.ToInt32(txtAddressNo
//invoke the delete method of the objec
MyAddressBook.Delete(AddressNo);
}
```

When this line is active press F11 to step into the middle layer...

Now press F10 until you get to the line of code “return true;”.


```

public Boolean Delete(Int32 AddressNo)
{
    ///this public function provides the functionality for the delete method

    //create an instance of the data connection class called MyDatabase
    clsDataConnection MyDatabase = new clsDataConnection();
    //add the AddressNo parameter passed to this function to the list of parameters to use in the database
    MyDatabase.AddParameter("@AddressNo", AddressNo);
    //execute the stored procedure in the database
    MyDatabase.Execute("sproc_tblAddress_Delete");
    //set the return value for the function
    return true;
}

```

Now look in the table to see if the record has been deleted.

	AddressNo	HouseNo	Street	Town	PostCode	CountyCode	DateAdded	Active
▶	1	1		Some Street	LE1 1BE	34	07/08/2013	True
	2	22		The Road	N19 6FF	48	07/08/2013	True
	3	33	The Mews	High Street	LE1 6FG	34	07/08/2013	True
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

If the record is still there and there or there were errors check that the name of the parameter and stored procedure in your middle layer code perfectly match the names in the stored procedure.

There is one important line of code in the function that we want to take a look at as it leads on to the next part of the module.

Notice the last line of code in the delete function...

```

    //set the return va
    return true;
}

```

What is this all about?

This line of code sets the return value for the function.

To show how this may be used we will make a slight modification to the presentation layer code like so...

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    //create an instance of the class clsAddress called ThisAddress
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //declare a variable to store the address numbver to delete
    Int32 AddressNo;
    //declare a boolean variable to record success of the delete operation
    Boolean Success;
    //copy the data from the interface to the RAM converting the data to Int32
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //invoke the delete method of the object passing it the data entered by the user and recording the outcome
    Success = MyAddressBook.Delete(AddressNo);
    //if the operation was successful display a message to the user
    if (Success == true)
    {
        lblAddressNo.Text = "You deleted record number " + AddressNo;
    }
    else //display an error message
    {
        lblAddressNo.Text = "There was a problem deleting the record";
    }
}
}
```

We have added three things.

1. A Boolean variable called Success to record the outcome of the operation. A Boolean data type may only store two values true or false.
2. We use the assignment operator to copy the return value of the function to this variable.
3. We have added an If statement that will display a message if the delete operation was successful.

Run the program and delete another record, you should see a message telling you that everything worked OK.

Now modify the code for your Delete function in the middle layer.

Change the return value of the function from true to false.

```
myDatabase.Execute( "PROC_DELETE_DELETE ",
    //set the return value for the function
    return false;
}
```

Run the program and delete another record. What happens?

You should see a message stating there was a problem. By changing the return value of a function we may send data back to the function that originally made the call.

Adding Some Simple Error Checking

Although we are not using it to its full extent the return value of a function may be used to report back some data to the function that called it. In this case it is a Boolean value but it can be pretty much anything we want it to be.

Change the return value of the method to default to true and have a look at this modified code for the method to make better use of the return value...

```

public Boolean Delete(Int32 AddressNo)
{
    ///this public function provides the functionality for the delete method

    try //try to delete the record
    {
        ///create an instance of the data connection class called MyDatabase
        clsDataConnection MyDatabase = new clsDataConnection();
        ///add the AddressNo parameter passed to this function to the list of parameters to use in the database
        MyDatabase.AddParameter("@AddressNo", AddressNo);
        ///execute the stored procedure in the database
        MyDatabase.Execute("spdoc_tblAddress_Delete");
        ///set the return value for the function
        return true;
    }
    catch //do this only if the code above failed for some reason
    {
        ///report back that there was an error
        return false;
    }
}

```

To trigger the error misspell the name of your stored procedure or the parameter passed via the AddParameter method.

By the end of this work you should have a fully linked three layer web application that allows you to delete records by entering data in the presentation layer.