Creating the Methods (Add, Edit Delete, Filter)

[Main Heading 1](#_Toc19265979)

[Creating the Methods 1](#_Toc19265980)

[Creating the Add Method 1](#_Toc19265981)

[Creating the Delete Method 12](#_Toc19265982)

[Creating the Update Method 18](#_Toc19265983)

[Creating the Post Code Filter 29](#_Toc19265984)

# Main Heading

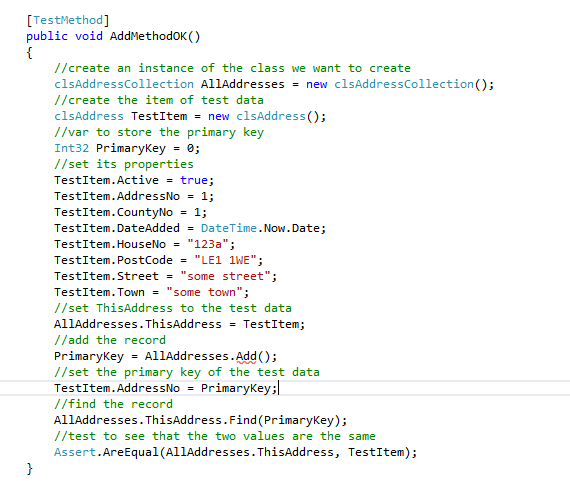
### Creating the Methods

The good news is that having created all of the tests for clsAddress we have a working find method already!

This means that we are in the position to press-on to testing Add, Update and Delete.

### Creating the Add Method

Here is the test method for Add…



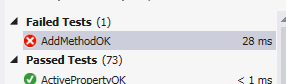
We have created rather more code here than usual. As we create more of the system and our confidence increases we have the option to make bigger steps.

Notice what we are trying to do here:

1. Create the collection
2. Create some test data
3. Use the test data to set the ThisAddress property
4. Add the record to the database retrieving the primary key of the new record
5. Find the record to check that it exists
6. Compare the data found with the original test data (they should be the same)!

Run the test and watch it fail.

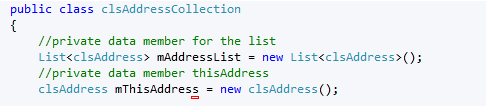
Create the Add method and the test will still fail!



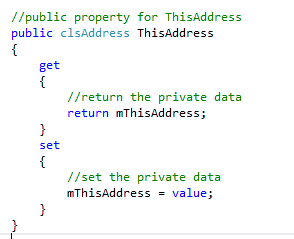
Why? It is because we don’t have any suitable code in the class to implement the method.

To fix this we will need to create a new private data member, modify the ThisAddress property and finally add some code to the Add method.

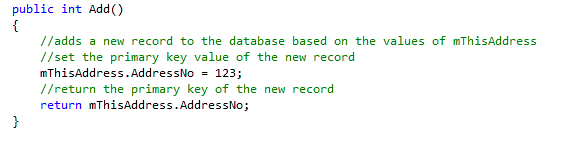
First let’s create the private data member at the top of the class…



Next we expose the private data member by modifying the public property like so…



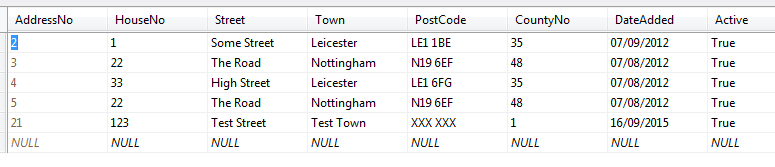
Lastly we put a fix in the Add method to force the test to pass…



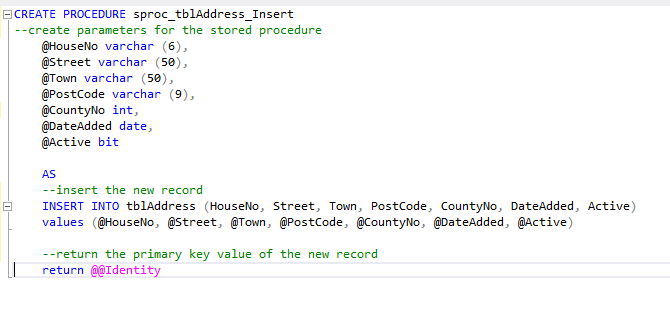
The good news is that it forces the test to pass. The bad news is that the code is pretty rubbish!

It would make more sense now to refine the code to make it actually work.

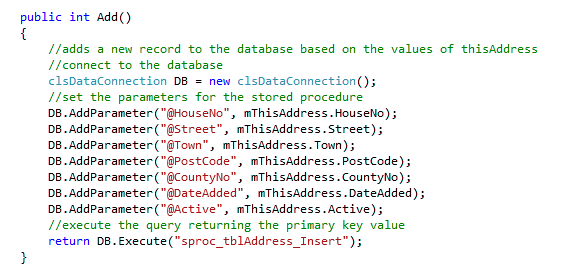
Assuming we have the following table…



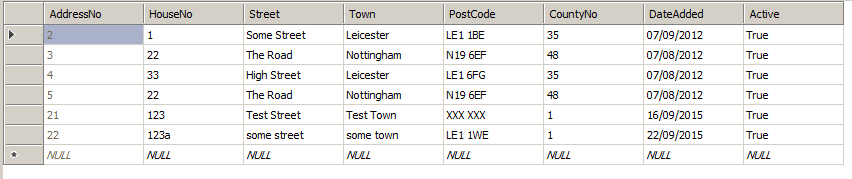
Along with the following stored procedure…



The amended function for Add should do the trick…

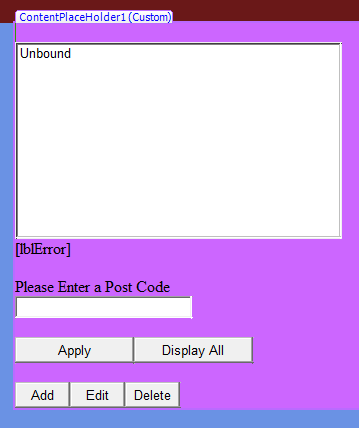


Lastly by examining the database we are able to see that a new record has been added…

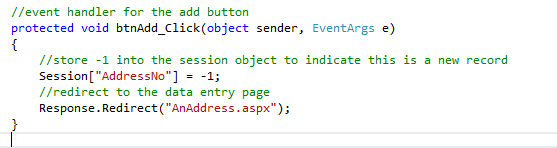


Having created the testing and the code for the Add method let’s add this to the presentation layer.

Open the form Default.aspx…

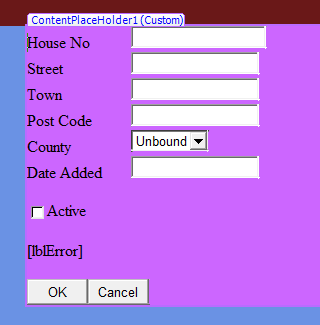


And access the event handler for the click event…



Notice how a value of -1 is placed in the session object on the server.

Once this value is set we then re direct to the page AnAddress.aspx…

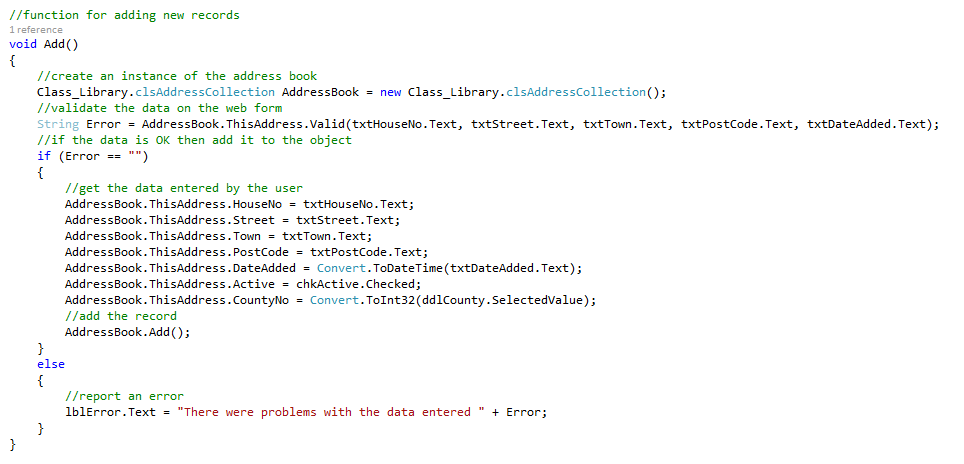


What we want to happen is that the user enters data into the fields on this form, presses OK and the data is added to the table via the middle layer classes.

We need to create a function which…

* Validates the data
* Captures the data
* Adds it to the database

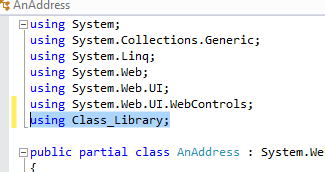
Create the following function inside AnAddress.aspx…



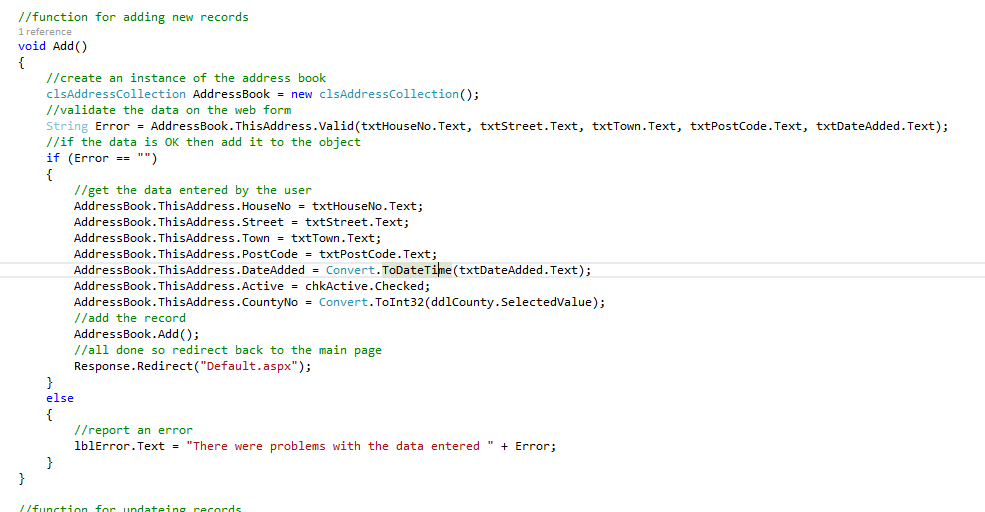
Notice the line of code…



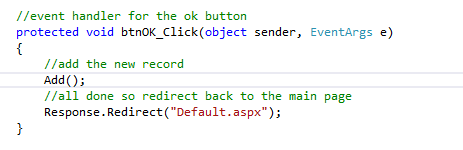
If you want to make this a bit more concise add a “using” to the top of the code like so…



In which case the function changes like so…



You will need to add a call to the click event handler for the OK button…

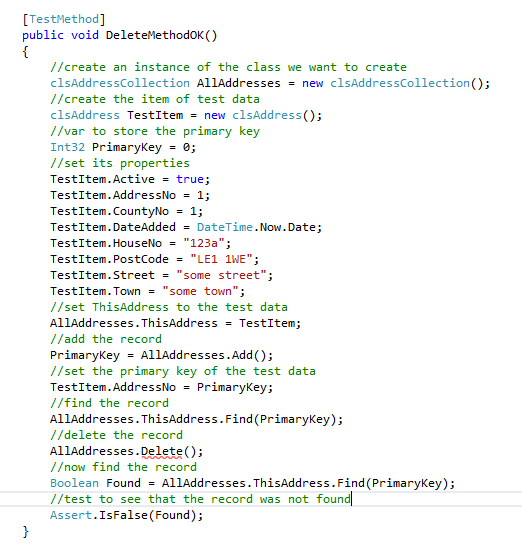


Again it is worth noting how quickly we were able to add the functionality to the presentation layer once we had the middle layer classes completed.

Also remember that having created the test framework we can be fairly sure that the functionality does what we think it should.

### Creating the Delete Method

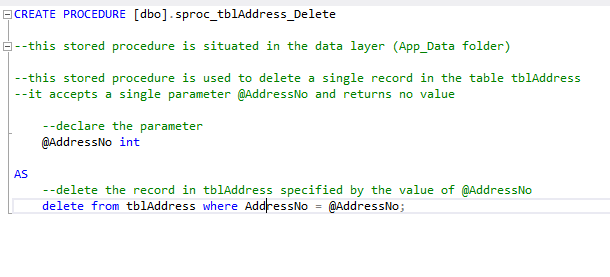
The next method we shall test is the Delete method…



It will fail due to no Delete method. Create the method and watch it fail for lack of suitable code.



Assuming we have a stored procedure like so…

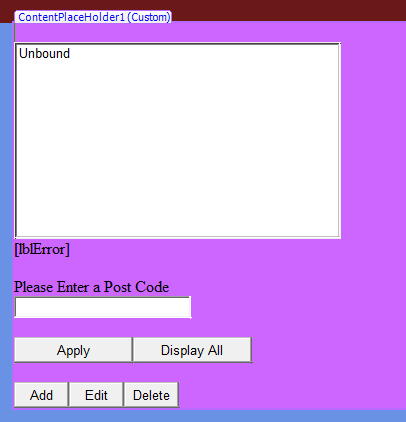


Here is the finished delete method…

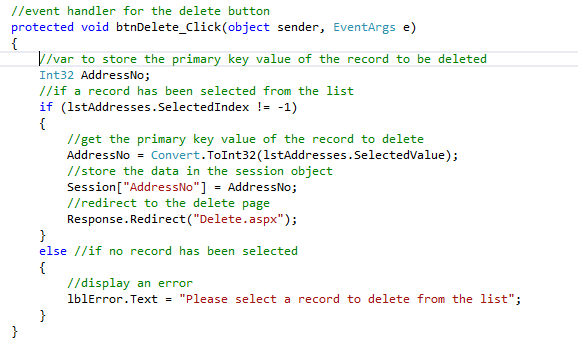


As before we will link the middle layer to the presentation layer.

From the form Default.aspx…



Access the click event handler for Delete…

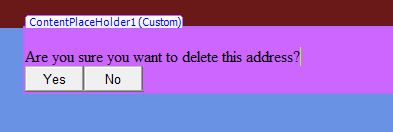


Notice how the code checks to see that an entry has been selected from the list.

Once this is OK we obtain the primary key value from the selected item of the list.

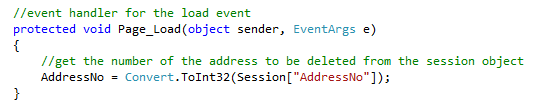
Next we pass the value of the primary key to the session object.

Lastly we redirect to the web form Delete.aspx…

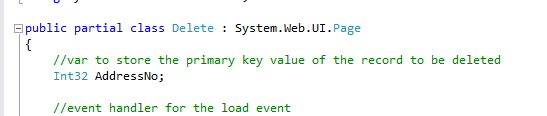


In order to obtain the primary key value of the record to delete we need to access the value stored in the session object.

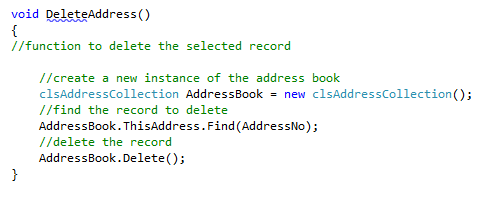
We will do this via the load event like so…



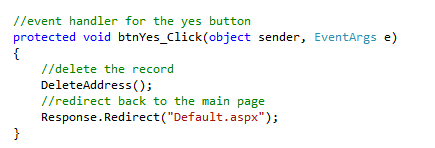
Notice how the variable AddressNo is declared at the top of the code giving it page level scope…



Here is the function…



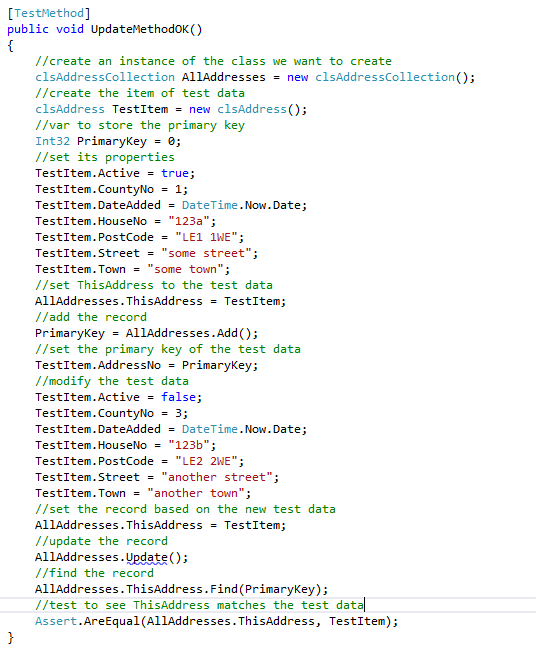
Here is the amended event handler…



### Creating the Update Method

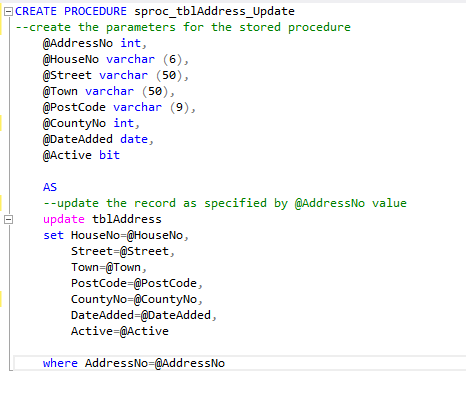
Lastly we shall create the Update method.

Here is the full test…

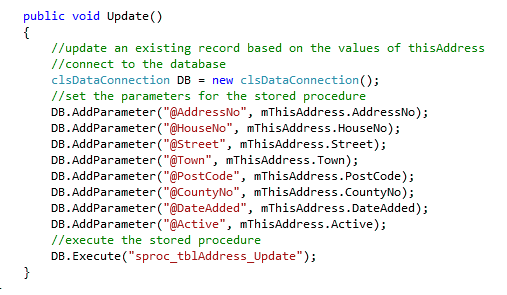


As usual it will fail due to lack of a stub for the method and a lack of code.

Here is the stored procedure we are using…

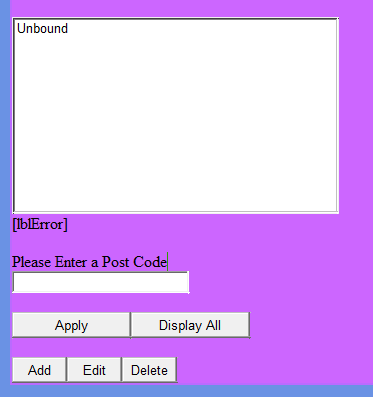


Here is the finished Update method…

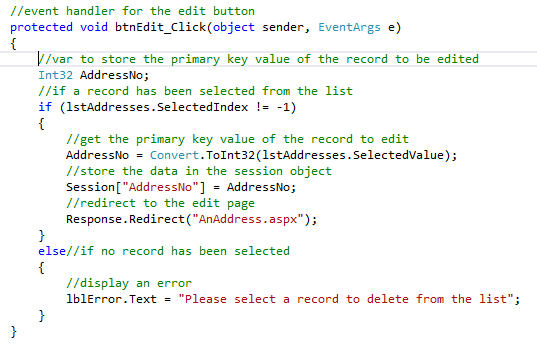


As before with Add and Delete we are now in a position to bolt the middle layer to the presentation layer.

In the form Default.aspx…



Access the click event handler for Edit…



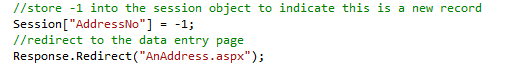
Note as with the Delete event handler we first check to see if the list has been selected.

Assuming it has we then place the primary key value into the session object.

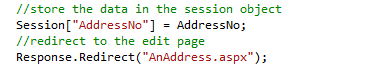
Then we redirect to the page AnAddress.aspx.

Compare the event handler operations for Add and Update.

For Add we place a -1 in the session object like so…

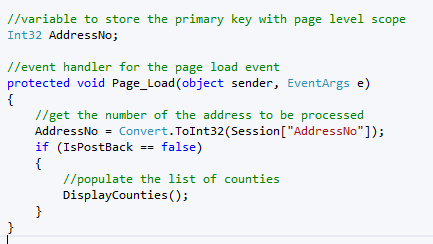


For Update we place the value of the primary key into the session object…



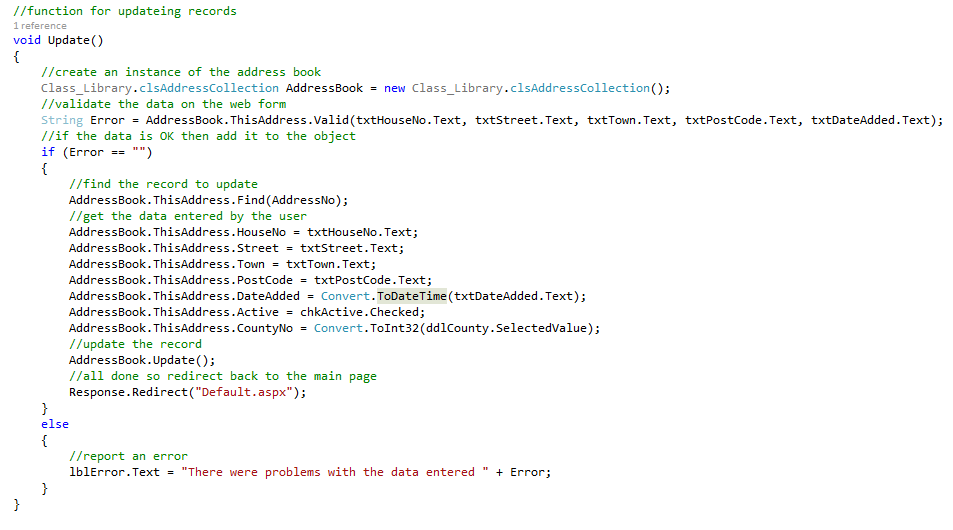
When we arrive in the page AnAddress.aspx we may use this to identify if the web form needs to add a new record or update an existing record.

The following code in the load event obtains the primary key value…

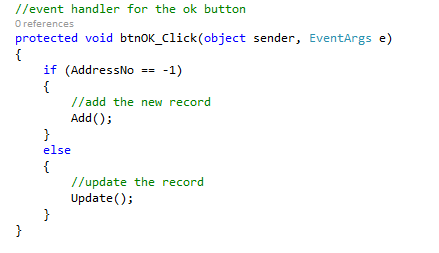


Notice how the AddressNo variable is declared with page level scope.

Here is the Update function…



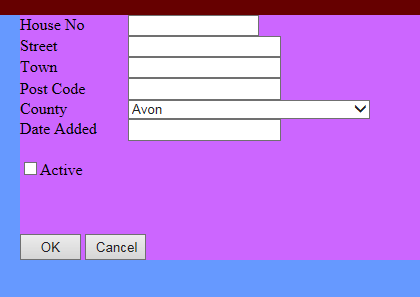
Here is the event handler for the OK button…



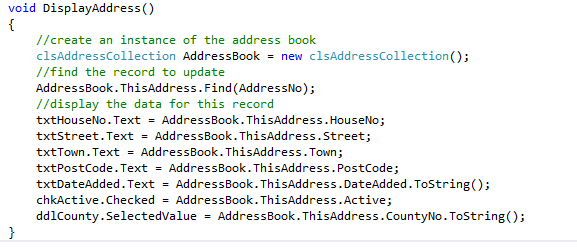
We are almost there!

We still have one last issue to address.

When the form AnAddress.aspx is displayed for editing a record it isn’t much good if it doesn’t display the existing data…

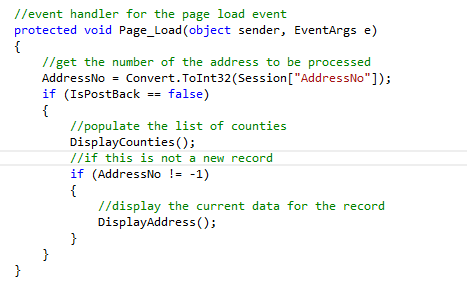


The following function solves this problem…



Also we only want to display the existing data when the form is first loaded, not once the user has entered some new data.

We will need to modify the load event like so…



You should now have a fully implemented set of classes with an associated test framework.

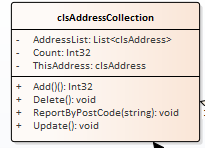
Are we done yet?

Not really there is still plenty of work to complete.

### Creating the Post Code Filter

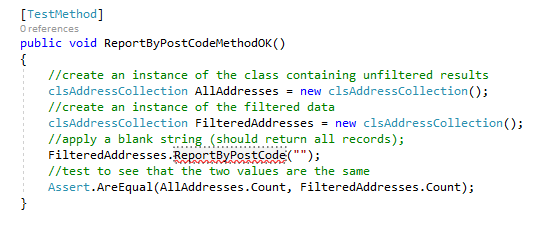
Filtering/reporting data is a really important part of any system. If we have 10,000 records we do not want to make the user trawl through them manually. There needs to be mechanisms to limiting the data and seeing only the records of interest.

The last method we will create is the method ReportByPostCode…



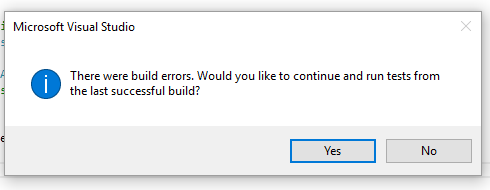
As with all previous examples it’s business as usual.

We need to start with a test to ensure that the method exists in the class like so…



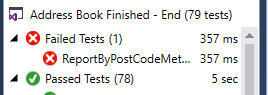
In this test we are applying a blank string to the filtered data. This filter should produce all results. By using a second instance of clsAddressCollection we may compare the two. A filter of blank string should produce the same count of records as the unfiltered results.

Run the tests and watch it fail.

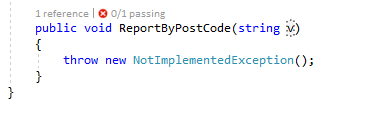


Now fix the test by generating the method stub.

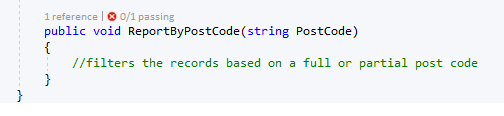
It should still fail…



Look at the function for the method and note the code you need to get rid of…

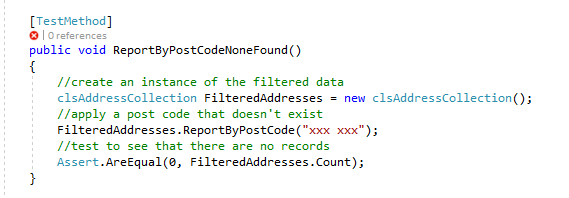


Lastly tidy up the parameter so that it makes more sense.



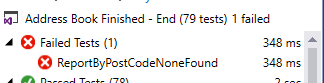
The test should now pass.

Having applied a test that looks for all records we shall apply a test that should produce no records.



(For this to work we need to make sure that the test data doesn’t contain a post code “xxx xxx”)

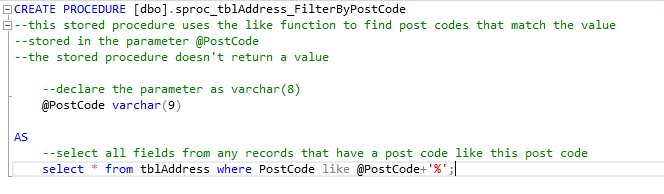
Run the test and watch it fail…



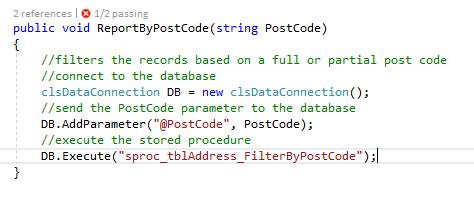
We now need to think about how to make the function work properly.

At this stage in the game we may as well cut to the chase.

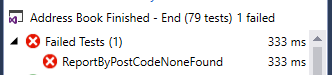
Assume we have a stored procedure called sproc\_tblAddress\_FilterByPostCode…



The following code should make the filter work…



Run the tests…



Watch the test fail again!

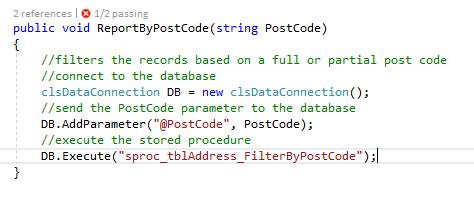
What is the problem now?

If we look at the code in the constructor that generates the collection we see the following…



Here we are populating the private array list mAddressList with the data from the data table in the DB object.

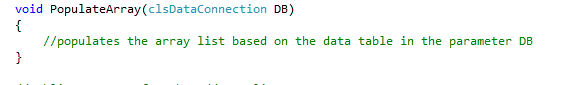
When we apply our post code filter…



The data in this DB object is not being sent to the private array list.

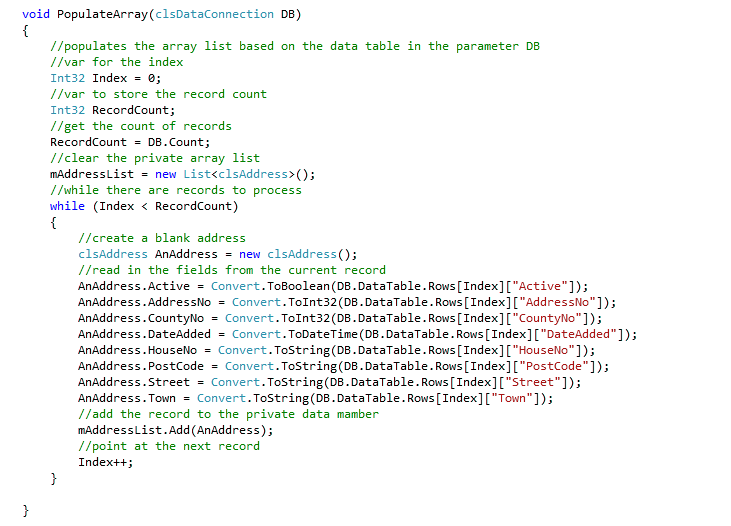
The neatest way to fix this is as follows.

Firstly create a new function called PopulateArray.



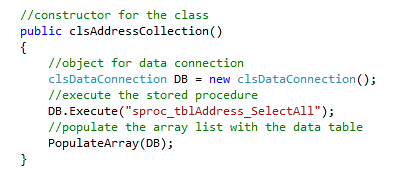
Notice how this function accepts a parameter called DB of type clsDataCollection.

Now add the code to the function like so…

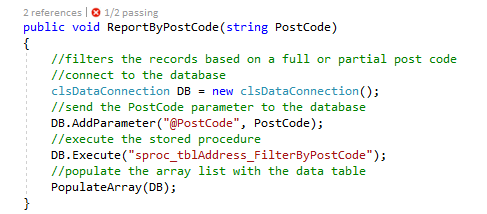


We now have a function that is dedicated to the task of copying whatever data is in a Data Connection to the private array list.

We need to modify the constructor like so…



And now modify the filter like so…

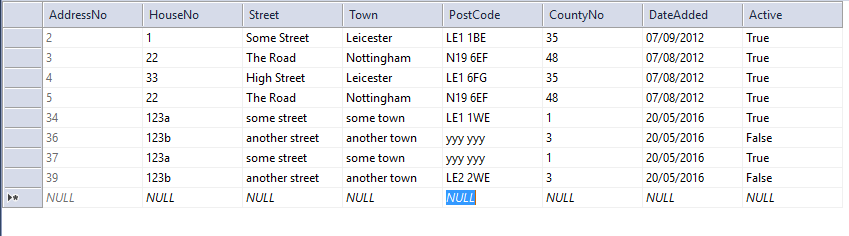


Since both function make use of the same code to populate the array list via the PopulateArray function the data should now be updated correctly.

The test should now pass!

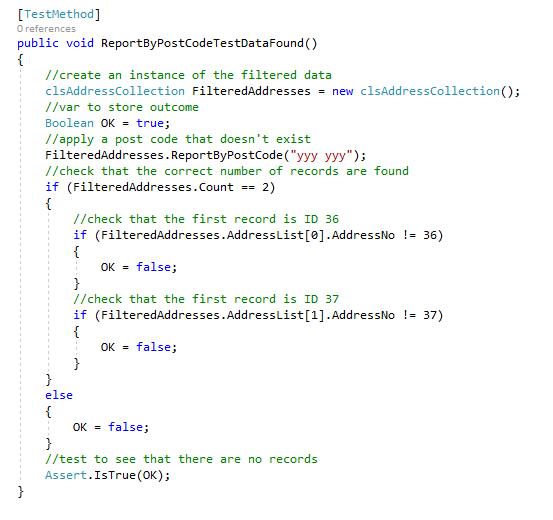
We are now able to establish that a blank filter is producing the correct number of records. We also know that an invalid filter produces zero records. However this doesn’t tell us if they are the correct records.

The final test to perform would be to create a couple of test records in the table that have the same post code.



If we filter on this data we should obtain these records and these two records only.

The following test makes a start on this…



This test only looks at the primary key values and doesn’t check the data of the other fields.

Having got this far there is still a lot to do. The filtering needs attaching to the presentation layer and you will also need to re-factor your code to make it as efficient as possible.