Creating the Methods (Add, Edit Delete, Filter)

Completing the Collection Class

[Completing the Collection Class 1](#_Toc34999353)

[Creating the Methods 1](#_Toc34999354)

[ThisAddress 1](#_Toc34999355)

[Creating the Add Method 3](#_Toc34999356)

[Creating the Delete Method 11](#_Toc34999357)

[Creating the Update Method 17](#_Toc34999358)

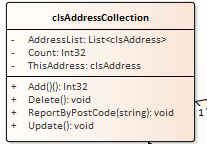
[Creating the Post Code Filter 24](#_Toc34999359)

[Modifying the Presentation Layer 32](#_Toc34999360)

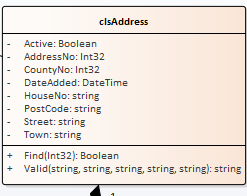
# Completing the Collection Class

## ThisAddress

At this point it is worth thinking about how the property ThisAddress is going to work and how it relates to the methods Add, Update and Delete.

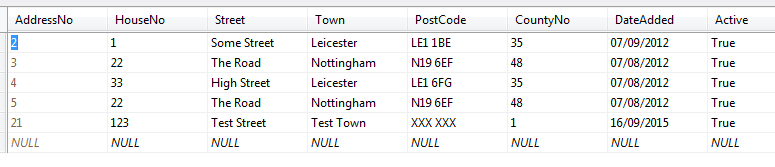


ThisAddress is an instance of clsAddress which we have already seen looks like this...



The find method allows us to search for a specific address based on its primary key value.

Assuming we have the following data...



The following code...

clsAddress SomeAddress = new clsAddress();

SomeAddress.Find(3);

Would locate the second record in the list setting the values of the properties of SomeAddress to match the data stored in the record.

We can use the find method of clsAddress via any instances of clsAddressCollection.

Compare the following code...

clsAddressCollection AllAddresses = new clsAddressCollection();

AllAddresses.ThisAddress.Find(3);

This code would achieve the same outcome as the previous code example.

Having the ThisAddress property allows us to point to any address we want to do something to, by searching on its primary key.

Once we are pointing at a specific record we may use Add, Update and Delete to change the data pointed to by ThisAddress.

So...

clsAddressCollection AllAddresses = new clsAddressCollection();

AllAddresses.ThisAddress.Find(3);

AllAddresses.Delete();

Would delete the address with the primary key value of 3.

Or...

clsAddressCollection AllAddresses = new clsAddressCollection();

AllAddresses.ThisAddress.Find(3);

AllAddresses.ThisAddress.HouseNo = "22";

AllAddresses.Update();

Would set the value of HouseNo for record 3 to "22".

Or...

clsAddressCollection AllAddresses = new clsAddressCollection();

AllAddresses.ThisAddress.Active = true;

AllAddresses.ThisAddress.CountyNo = 1;

AllAddresses.ThisAddress.DateAdded = DateTime.Now.Date;

AllAddresses.ThisAddress.HouseNo = "22";

AllAddresses.ThisAddress.PostCode = "LE1 1WE";

AllAddresses.ThisAddress.Street = "Some Street";

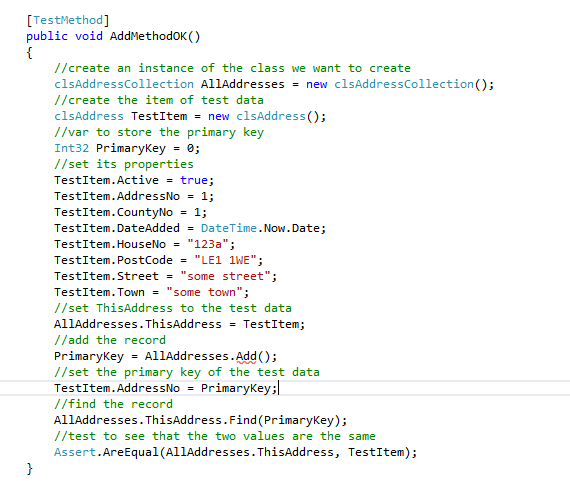
AllAddresses.ThisAddress.Town = "Some Town";

AllAddresses.Add();

Would add a new record based on the data we assigned to ThisAddress. (Note in this example Find isn’t required.)

### Creating the Add Method

Here is the test method for Add…



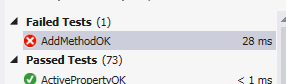
We have created rather more code here than usual. As we create more of the system and our confidence increases we have the option to make bigger steps.

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.

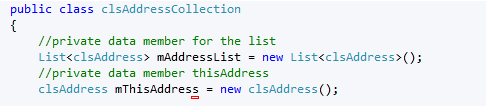
Currently the auto generated Add method in clsAddressCollection will look like this



To fix this we will need to make a few changes to the code.

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

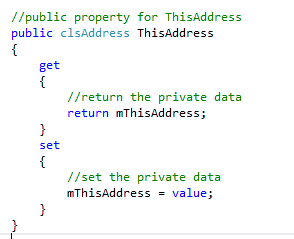
First let’s create the private data member for ThisAddress at the top of the class…



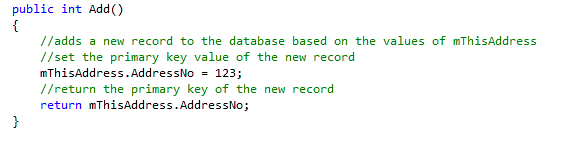
Next we expose the private data member by modifying the public property from this…



To this…



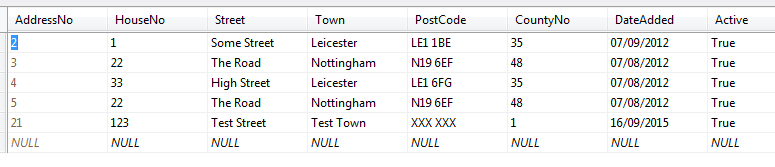
Lastly we put a fix in the Add method to force the test to pass…



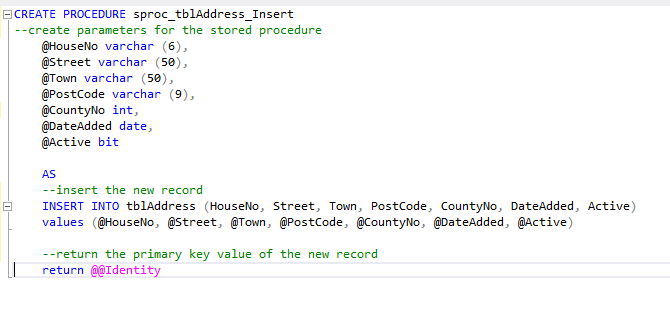
The good news is that it forces the test to pass. The bad news is that the code is pretty rubbish!

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

Assuming we have the following table…

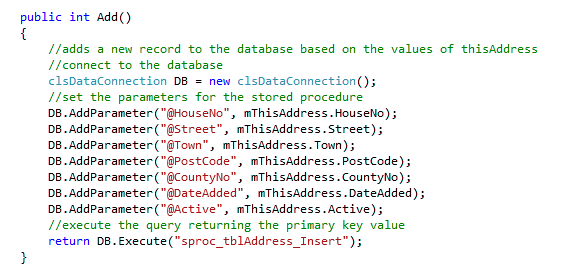


Along with the following stored procedure…

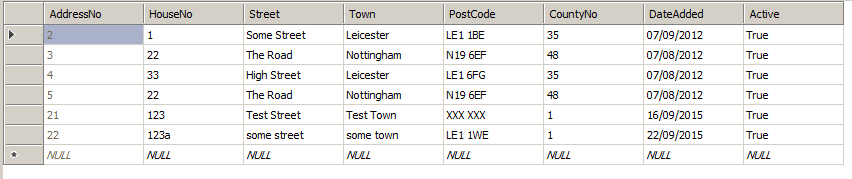


It is important to remember when adding a new record that the primary key is read-only and generated by the system. Notice how I do not include AddressNo, my primary key in the above SQL.

The amended function for Add should do the trick…

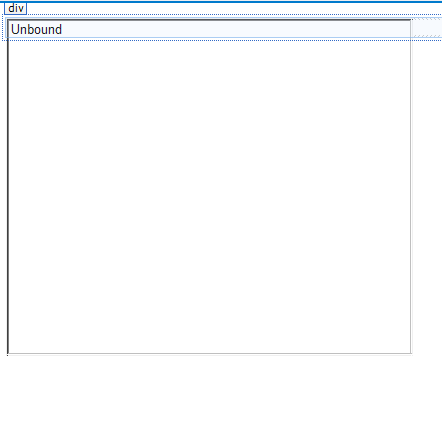


Lastly by examining the database we are able to see that a new record has been added…



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

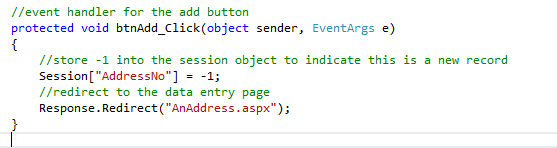
Open the form AddressList.aspx we created in the last section of work…



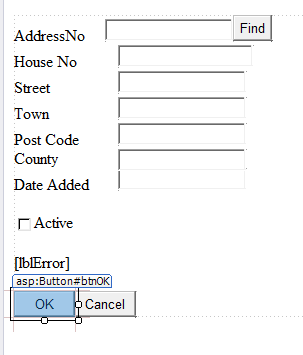
Below the list add a new button for Add with a label of btnAdd like so…



And access the event handler for the click event…



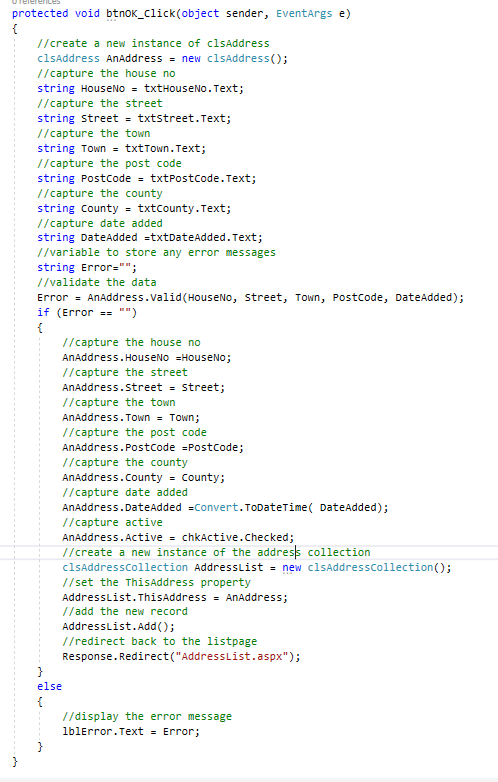
Notice how a value of -1 is placed in the session object on the server. What we need to do is send a message to the editing form AnAddress.aspx that this an add operation. That way when we press OK on the editing form…



The system knows to add the inputted data as a new record.

(Make a mental note of this as we shall return to it when we implement delete and update)

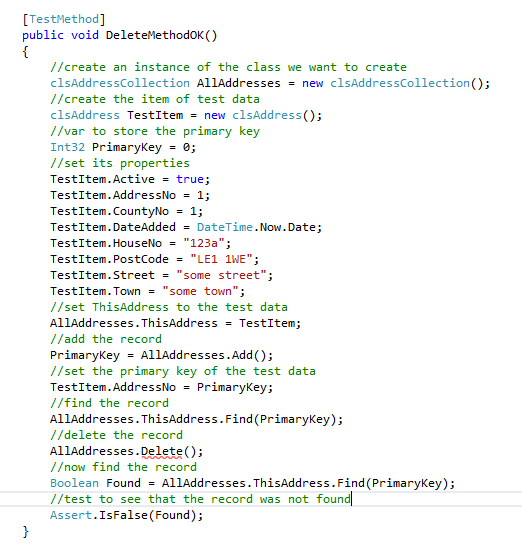
Next, modify the event handler in AnAddress.aspx for the OK button like so…



Run the program to see if you can add new records using the interface.

### Creating the Delete Method

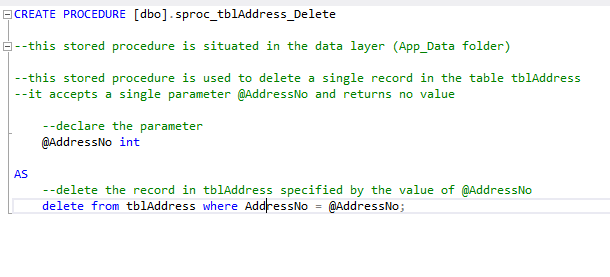
The next method we shall test is the Delete method…



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…

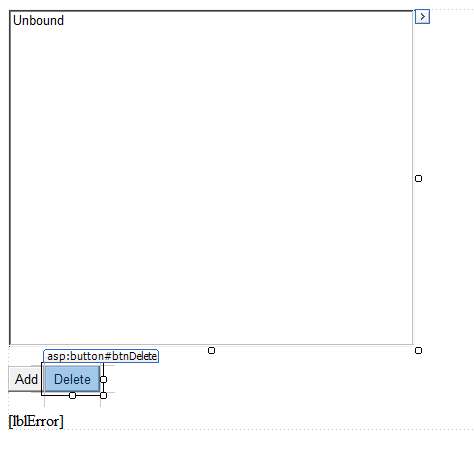


As before we will link the middle layer to the presentation layer.

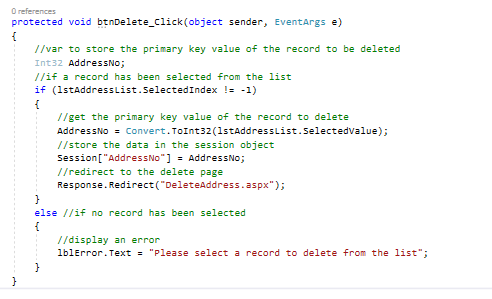
From the form Default.aspx…



Add a new button for delete called btnDelete and a blank label called lblError like so…



Access the click event handler for Delete and add the following code…



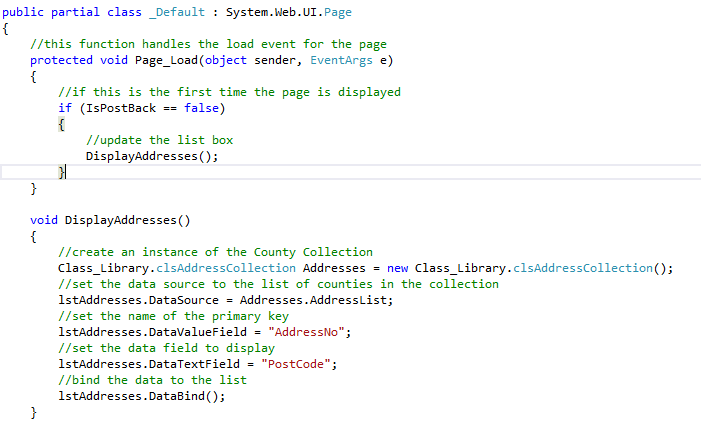
There are few things in this code worth pointing out.

lstAddressList.SelectedIndex != -1 allows us to make sure that an entry on the list has been clicked.

Each item in the list has an index from 0 to N. If the first item is clicked accessing SelectedIndex will produce a value of 0. If nothing has been selected then the SelectedIndex will be 01 so we need to tell the user to click something.

Notice how we access the primary key value via the SelectedValue property.

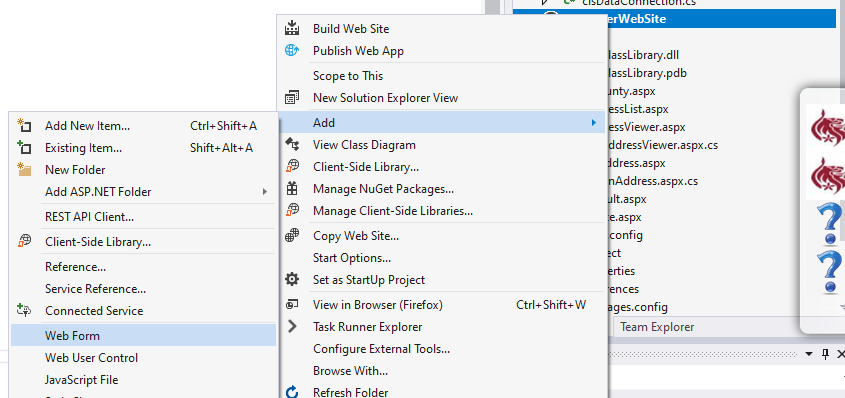
When we set up the code to display the list from the collection we add the following…



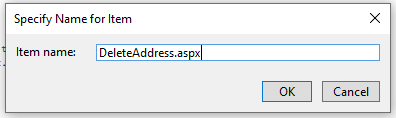
By setting the DataValueText we tell the list which property to display to the user.

By setting the DataValueField we tell the list which property contains the primary key value.

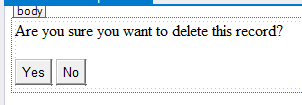
Next you need to create a new web form called DeleteAddress.aspx…



Set the name of the form to DeleteAddress.aspx

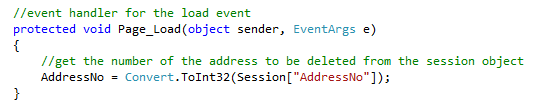


Set up the web form like so…

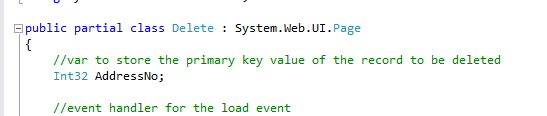


In order to obtain the primary key value of the record to delete we need to access the value stored in the session object.

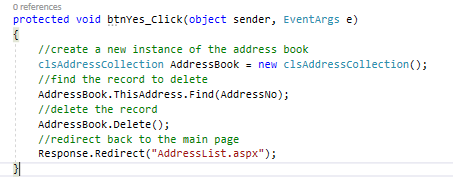
We will do this via the load event of DeleteAddress.aspx like so…



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



Modify the event handler of the Yes button like so…

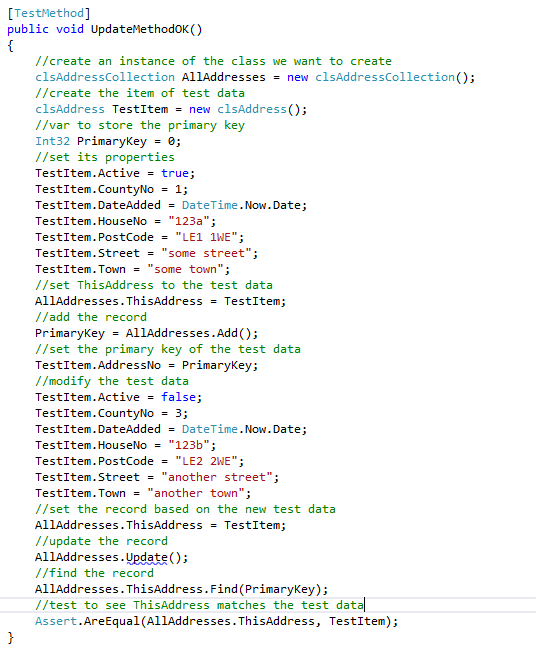


Again test the functionality by running the web site.

### Creating the Update Method

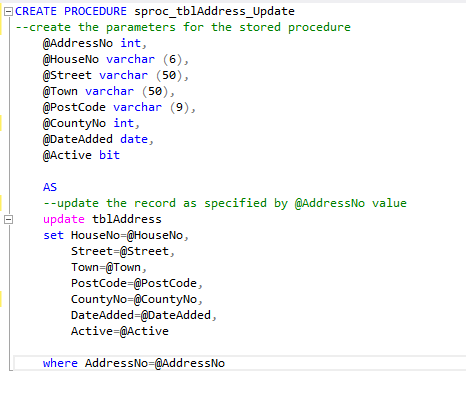
Lastly we shall create the Update method.

Here is the full test…



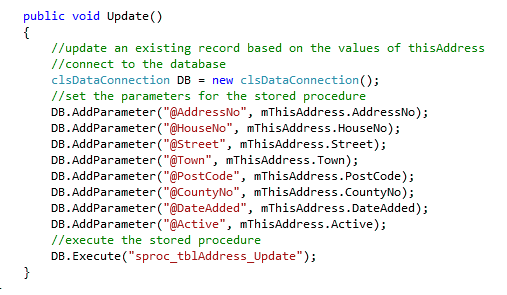
As usual it will fail due to lack of a stub for the method and a lack of code.

Here is the stored procedure we are using…



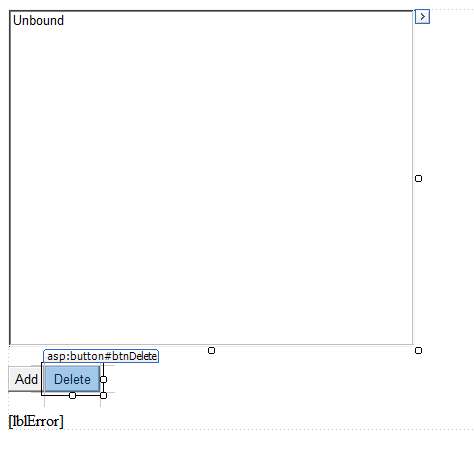
Worth pointing out that we do not write to the primary key in the “set” operation. Remember the primary key is read-only.

Here is the finished Update method…

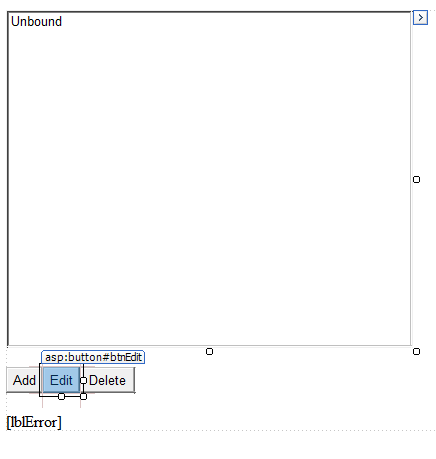


As before with Add and Delete we are now able to bolt the middle layer to the presentation layer.

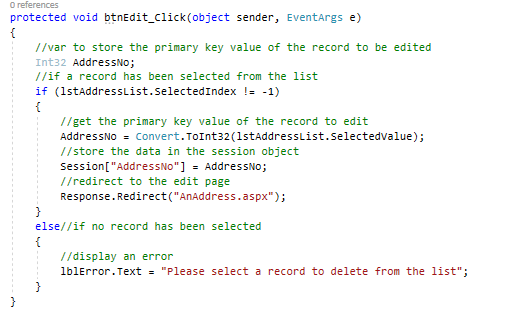
In the form Default.aspx…



Add a new button for Edit called btnEdit like so…



Access the click event handler for Edit…



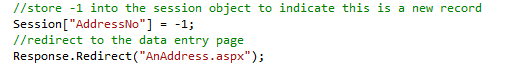
Note as with the Delete event handler we first check to see if the list has been selected and pick up the primary key value via the SelectedValue property of the list box.

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

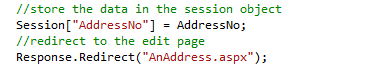
Then we redirect to the page AnAddress.aspx.

Compare the event handler operations for Add and Update.

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

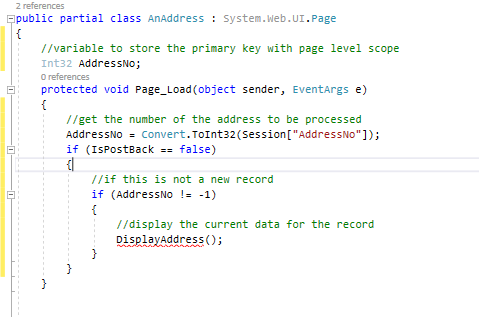


For Update we place the value of the primary key into the session object…



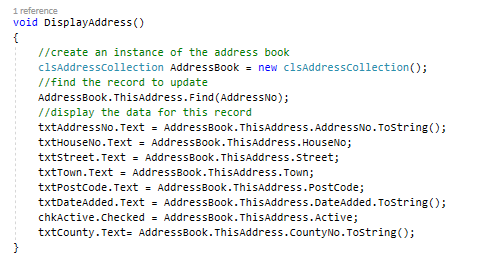
When we arrive in the page AnAddress.aspx we may use this to identify if the web form needs to add a new record or update an existing record.

The following code in the load event of AnAddress.aspx obtains this primary key value…

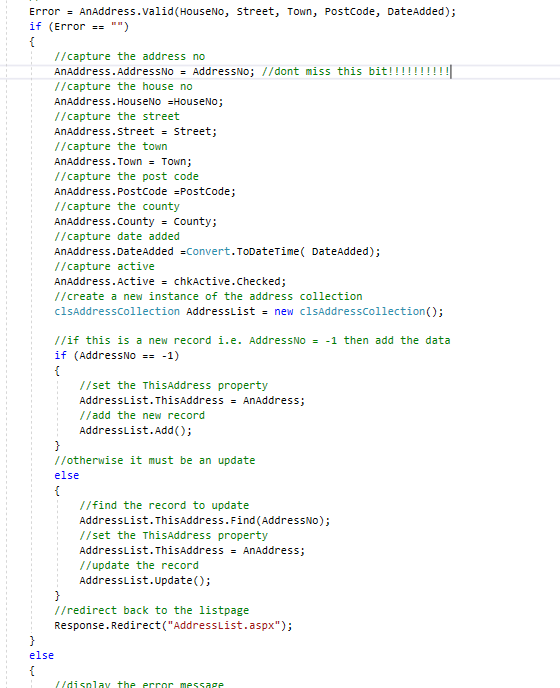


Notice how the AddressNo variable is declared with page level scope.

If we are to edit existing data to the user we need to display that data on the web form. The function DisplayAddress does this…



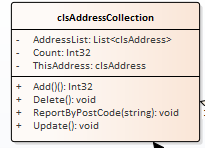
Finally, we need to update if statement in the event handler for the OK Button…



### Creating the Post Code Filter

Filtering/reporting data is an 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.

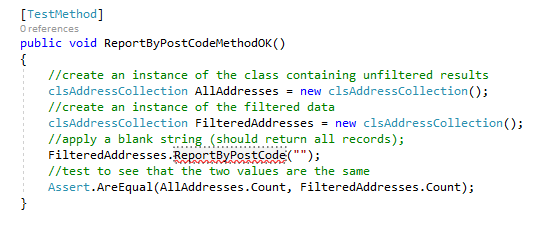
The last method we will create is the method ReportByPostCode…



In creating your own filter option, you will need to decide on a suitable field to filter on. You can in theory use any data type but at this stage it is best to select a field in your table that is of string data type.

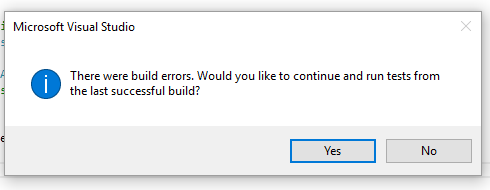
As with all previous examples it’s business as usual.

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



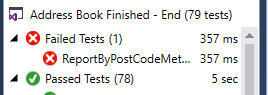
In this test we are applying a blank string to the filtered data. This filter should produce all results. By using a second instance of clsAddressCollection we may compare the two. A filter of blank string should produce the same count of records as the unfiltered results.

Run the tests and watch it fail.

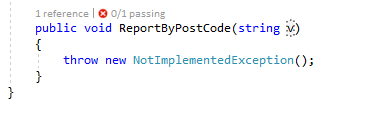


Now fix the test by generating the method stub.

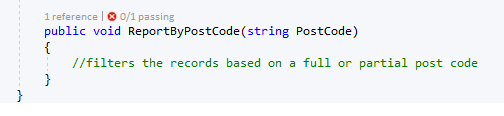
It should still fail…



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

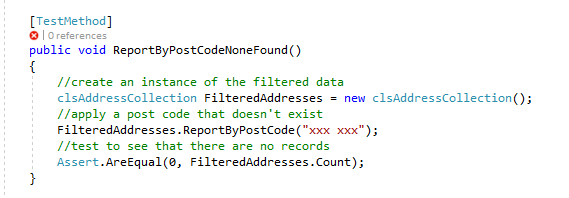


Lastly tidy up the parameter so that it makes more sense.



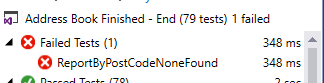
The test should now pass.

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”)

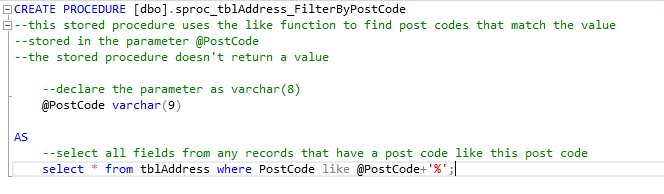
Run the test and watch it fail…



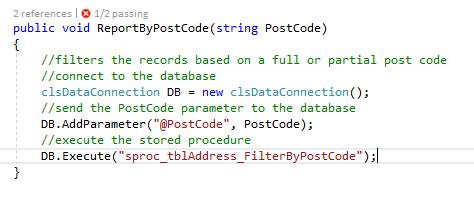
We now need to think about how to make the function work properly.

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

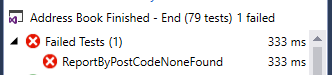
Assume we have a stored procedure called sproc\_tblAddress\_FilterByPostCode…



The following code should make the filter work…



Run the tests…



Watch the test fail again!

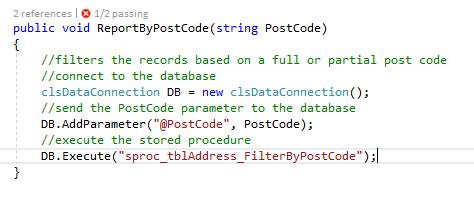
What is the problem now?

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



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

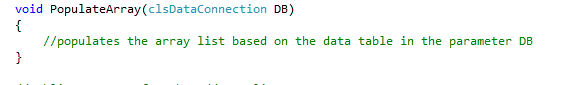
When we apply our post code filter…



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

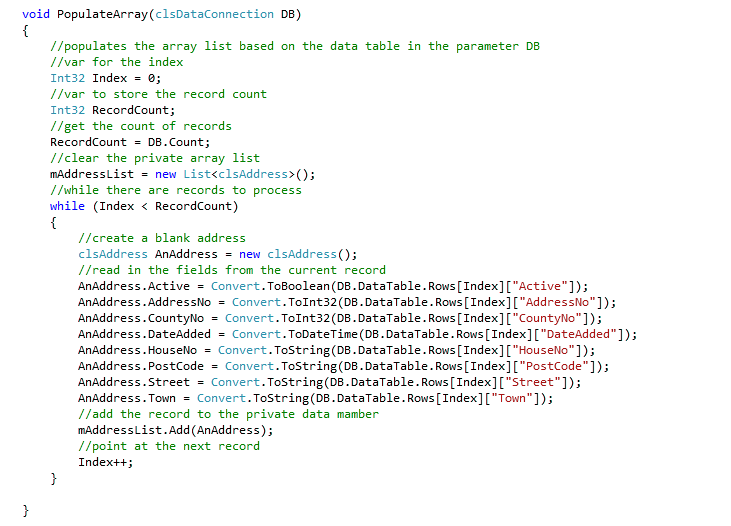
The neatest way to fix this is as follows.

Firstly create a new function called PopulateArray.



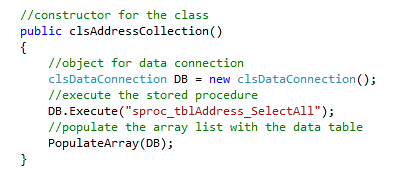
Notice how this function accepts a parameter called DB of type clsDataCollection.

Now add the code to the function like so…

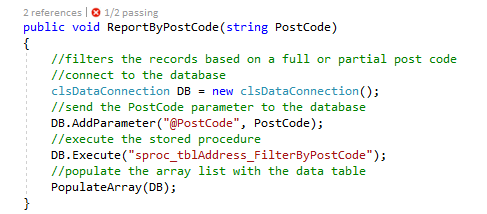


We now have a function that is dedicated to the task of copying whatever data is in a Data Connection to the private array list.

We need to modify the constructor for clsAddressCollection like so…



And now modify the filter like so…

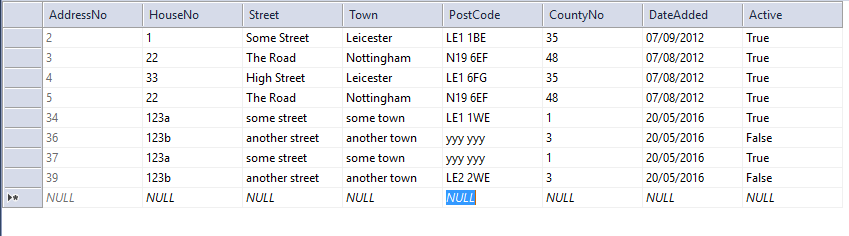


Since both function make use of the same code to populate the array list via the PopulateArray function the data should now be updated correctly.

The test should now pass!

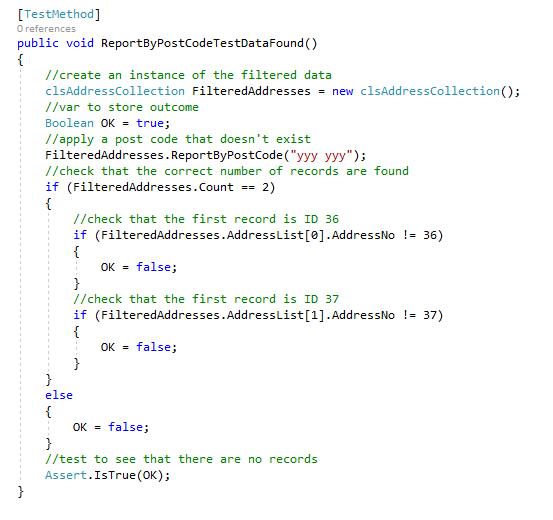
We are now able to establish that a blank filter is producing the correct number of records. We also know that an invalid filter produces zero records. However this doesn’t tell us if they are the correct records.

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…

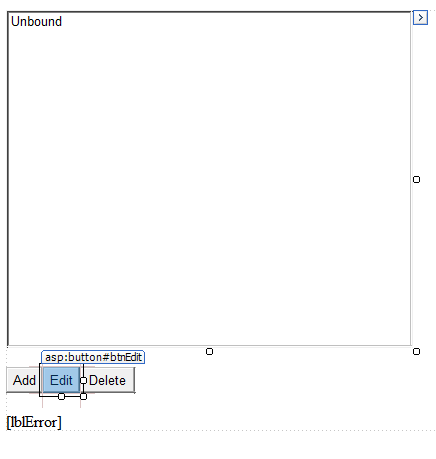


(You will need to set this test up so that it matches your own test data.)

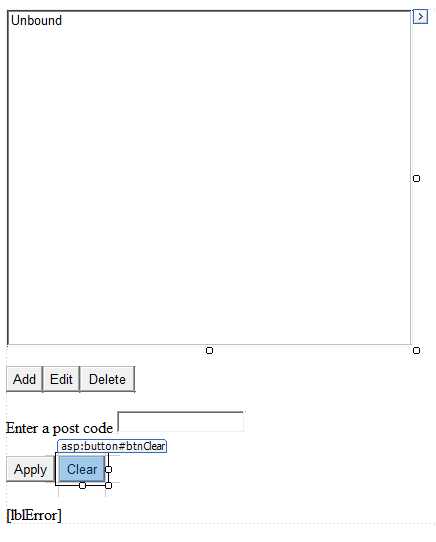
## Modifying the Presentation Layer

Finally let’s add this functionality to our presentation layer.

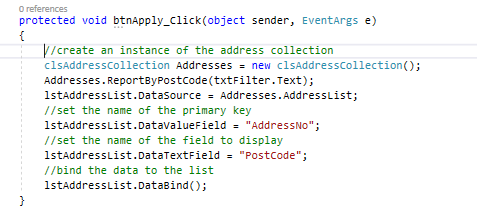
Open AddressList.aspx…



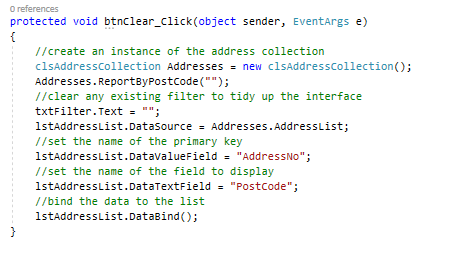
And modify it like so…



To the Apply button add the following code…



To the Clear button add the following code…



If you now enter a partial string on the interface it should apply pattern matching to display only selected records…

