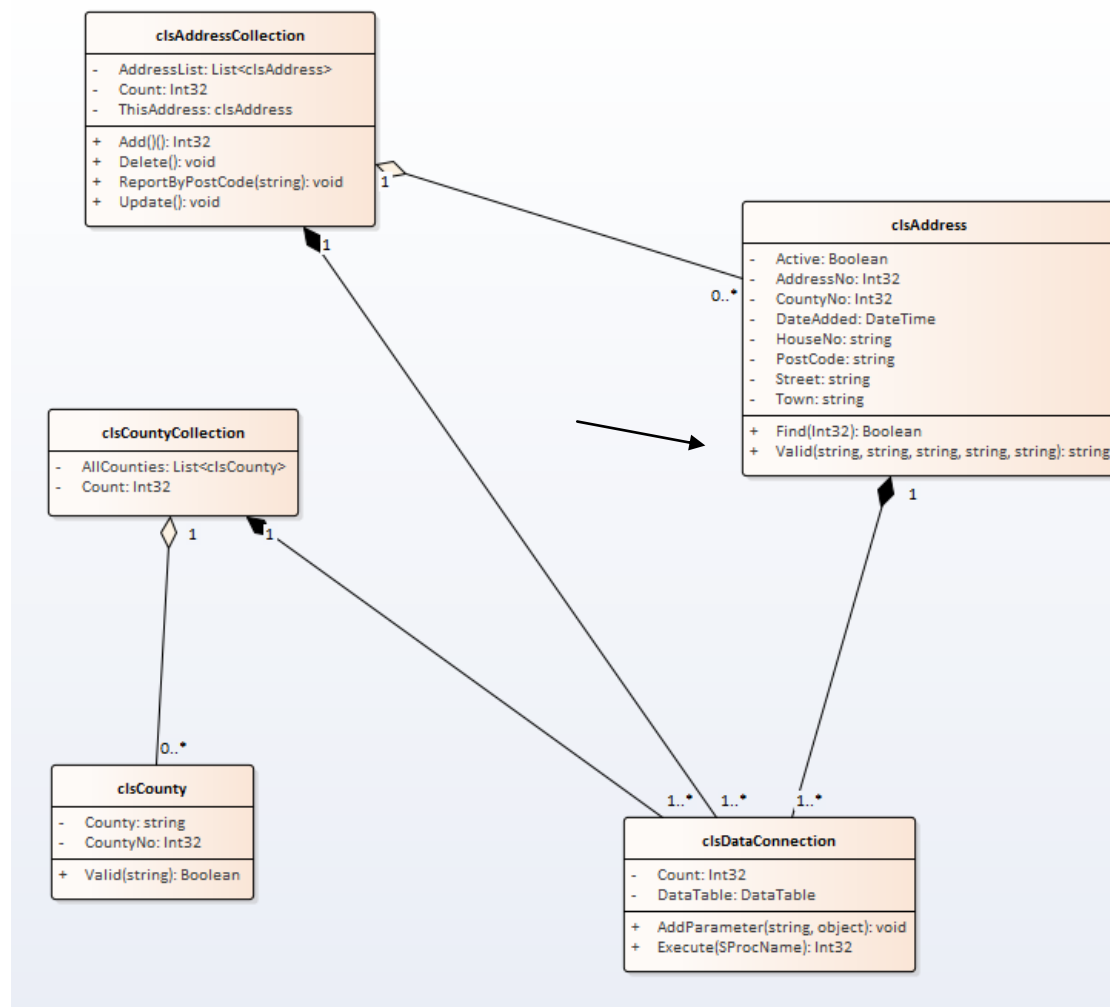


Creating the Validation Method

Having created your test plan you need to create code that ensures your application passes your test cases. The code will require several If statements contained within a suitable function. The logic we shall use is as follows...

1. Declare a string variable to store the error message (if any)
2. Initialise the variable with a blank string
3. Test a unit of data to see if it is OK
If it isn't OK concatenate an error into the string variable
If it is OK then go on to the next test
4. Perform all other tests using the same logic as 3
5. Once all tests are completed return the value of the string variable as the return value of the error message

The validation function will be created as a method in the class clsAddressPage. This will produce a method.



Creating the Function

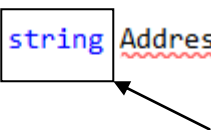
The first step is to create the stub for the function so that we have a place to put the validation code. Inside your class definition for clsAddress create the following public function...

```
public string AddressValid(string AddressNo,  
                             string HouseNo,  
                             string Street,  
                             string Town,  
                             string PostCode,  
                             string DateAdded)  
{  
}
```

Things to note...

- We have split the function definition over several lines. We don't need to use this trick but it does help to make it easier to read.
- We have a parameter for each item we want to test. It is important to point out also that each parameter has a data type of string. The reason for this is that string will accept data of any data type. Since we are passing data to this function that is yet to be validated we cannot make any assumptions about it being correct or not. For example DateAdded should be a date, however if the user enters an invalid date we cannot pass the data to the function if we make this parameter DateTime data type.
- The return data type of the function is string as it will contain the text of any error message.

```
public string AddressValid(string AddressNo,  
                             string HouseNo,  
                             string Street,  
                             string Town,  
                             string PostCode,  
                             string DateAdded)  
{  
}
```



In this case the function will return either a blank string if all is OK with the data or the error message if there is a problem.

- Lastly notice that the name of the function is underlined in red.

```
: AddressValid(:  
:               :
```

- This is because we haven't set up the function to return a value yet, we shall do that now.

Setting up the Return Value of the Function

As we have mentioned several times already a function returns a value. To set this up we need to complete the following steps...

1. Declare a string variable to store the error message (if any)
2. Initialise the variable with a blank string
5. Once all tests are completed return the value of the string variable as the return value of the error message

Firstly declare a variable to store any error message...

```
public string AddressValid(string AddressNo,  
                           string HouseNo,  
                           string Street,  
                           string Town,  
                           string PostCode,  
                           string DateAdded)  
{  
    string ErrorMsg;  
}
```

Next initialise the variable with a blank string...

```
public string AddressValid(string AddressNo,  
                           string HouseNo,  
                           string Street,  
                           string Town,  
                           string PostCode,  
                           string DateAdded)  
{  
    string ErrorMsg;  
    ErrorMsg = "";  
}
```

Lastly return the value of the ErrorMsg variable...

```

public string AddressValid(string AddressNo,
                           string HouseNo,
                           string Street,
                           string Town,
                           string PostCode,
                           string DateAdded)
{
    string ErrorMsg;
    ErrorMsg = "";

    |

    return ErrorMsg;
}

```

Notice we have created a few blank lines so that we have a place to create our If statements.

The next step is to create the first of probably many If statements.

The first value we shall check is the house number.

Last week we made a start on the test plan for this data which looked something like this...

Description of Item to Be Tested:

This field stores the house number of the address. This may be a number or it could contain a letter e.g. 33B so the data type is string. The field must be no more than six characters and may not be blank.			
Test Type	Test Data	Expected Result	Actual Result
Extreme Min	Not applicable	NA	
Min -1	Blank	Display an error message stating required field.	
Min (Boundary)	1 character		
Min +1	2 characters	Should accept the data	
Max -1	12345	Should accept the data	
Max (Boundary)	123456	Should accept the data	
Max +1	1234567	Should display an error message	
Mid	123	Should be ok	
Extreme Max	1234567890123456789123456789	Should fail	

	0123456789 1234567890123456789123456789 0123456789 Etc..		
Invalid data type	NA		
Other tests	NA		

The important items to note in the case of this data are the boundaries of the data. In this case the value must have at least one character and may not be more than six characters.

To test this we would use the built in string method Length.

Length

Length inspects the data passed as a parameter and counts the number of characters. The return value of the method is the number of characters in the string.

```
//var to store the number of characters
Int32 Chars;
//var to store some text (hello)
string SomeText = "hello";
//get the length of the string
Chars = SomeText.Length;
```

In the above example the variable Chars would be assigned the value 5.

The If statement could be constructed in a number of ways, we shall opt for the following structure...

```
public string AddressValid(string AddressNo,
                           string HouseNo,
                           string Street,
                           string Town,
                           string PostCode,
                           string DateAdded)
{
    string ErrorMsg;
    ErrorMsg = "";

    if (HouseNo.Length < 1 | HouseNo.Length > 6)
    {

    }

    return ErrorMsg;
}
```

The logic for some of the If statements can be quite fiddly especially when you are getting started on this kind of thing. Expect to get this wrong while you are learning.

The logic above states if the house no has less than < one character OR more than > five characters generate an error.

The OR operator in C# is the | symbol.

The next step is to generate the error using concatenation...

```
public string AddressValid(string AddressNo,
                           string HouseNo,
                           string Street,
                           string Town,
                           string PostCode,
                           string DateAdded)
{
    ///this function is used to validate the data in a new address
    ///it accepts six parameters and returns a string containing the text of the errors (if any)
    ///otherwise of no errors it returns a blank string
    string ErrorMsg; //var to store any error message
    ErrorMsg = ""; //initialise the var with a blank string
    ///if the string is less than 1 char or more than 6
    if (HouseNo.Length < 1 | HouseNo.Length > 6)
    {
        ///record an error
        ErrorMsg = ErrorMsg + "The house number must be between 1 and 6 characters";
    }
    ///return any error messages generated
    return ErrorMsg;
}
```

(We have also added a few comments for good measure.)

What would be a really good idea at this stage is to test the function to see if it works as we think it should.

Testing the Function

To test the function we need to modify the event handler for the OK button on the data entry form AnAddress.aspx.

Go to design view and double click on the OK button to access the event handler like so...

```
protected void btnOK_Click(object sender, EventArgs e)
{
    //redirect back to the main page
    Response.Redirect("Default.aspx");
}
```

To test the validation function we will use the following code...

```
protected void btnOK_Click(object sender, EventArgs e)
{
    //create an instance of the address page class
    clsAddress ThisAddress = new clsAddress();
    //var to store any error messages
    string ErrorMessage;
    //test the data on the web form
    ErrorMessage = ThisAddress.AddressValid(txtAddressNo.Text,
                                            txtHouseNo.Text,
                                            txtStreet.Text,
                                            txtTown.Text,
                                            txtPostCode.Text,
                                            txtDateAdded.Text);

    if (ErrorMessage == "")
    {
        //do something with the data - insert or update
        //redirect back to the main page
        Response.Redirect("Default.aspx");
    }
    else
    {
        //display the error message
        lblError.Text = ErrorMessage;
    }
}
```

To test this place a break point in the function (F9) and then run the page. Enter some dummy data and see (using F10) if the validation works or not.

Now that we have a mechanism for checking if the logic for the validation is ok we may now go on and validate the rest of the data entered.

Here is a bit more code to get you started...

```
//var to store the error message
string ErrMsg = "";
//check the min length of the house no
if (HouseNo.Length == 0)
{
    //set the error message
    ErrMsg = ErrMsg + "House no is blank. ";
}
//check the max length of the house no
if (HouseNo.Length > 6)
{
    //set the error message
    ErrMsg = ErrMsg + "House no must be less than 6 characters. ";
}
//check the min length of the street
if (Street.Length == 0)
{
    //set the error message
    ErrMsg = ErrMsg + "Street is blank. ";
}
//check the max length of the street
if (Street.Length > 50)
{
    //set the error message
    ErrMsg = ErrMsg + "Street must be less than 50 characters. ";
}
//check the min length for the town
if (Town.Length == 0)
{
    //set the error message
    ErrMsg = ErrMsg + "Town is blank. ";
}
//check the max length for the town
if (Town.Length > 50)
{
    //set the error message
    ErrMsg = ErrMsg + "Town must be less than 50 characters. ";
}
//check the min length for the post code
if (PostCode.Length == 0)
{
    //set the error message
    ErrMsg = ErrMsg + "Post Code is blank. ";
}
//check the max length for the post code
if (PostCode.Length > 9)
```



```

{
    //set the error message
    ErrMsg = ErrMsg + "Post Code must be less than 9 characters. ";
}
//test if the date is valid
try//try the operation
{
    //var to store the date
    DateTime Temp;
    //assign the date to the temporary var
    Temp = Convert.ToDateTime(DateAdded);
}
catch//if it failed report an error
{
    //set the error message
    ErrMsg = ErrMsg + "Date added is not valid. ";
}
//if there were no errors
if (ErrMsg == "")
{
    //return a blank string
    return "";
}
else//otherwise
{
    //return the errors string value
    return "There were the following errors : " + ErrMsg;
}

```

Once you think that you have created a validation function that works you need to enter the test data from the test plan to see if you get the expected error messages.