

Populating the List Box

We have covered a lot of ground in the module so far.

The last part of this stage of the development is actually seeing the data appear in the list box on the presentation layer.

So far you should have a function called AddressList which looks something like this...

```
0 references
public void AddressList()
{
    //create an array list of type clsAddressPage
    List<clsAddress> mAddressList = new List<clsAddress>();
    //var to store the count of records
    Int32 RecordCount;
    //var to store the index for the loop
    Int32 Index = 0;
    //create a connection to the database
    clsDataConnection dBConnection = new clsDataConnection();
    //send a post code filter to the query
    dBConnection.AddParameter("@PostCode", "");
    //execute the query
    dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
    //get the count of records
    RecordCount = dBConnection.Count;
    //keep looping till all records are processed
    while (Index < RecordCount)
    {
        //create a blank address page
        clsAddress NewAddress = new clsAddress();
        //copy the data from the table to the RAM
        NewAddress.AddressNo = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["AddressNo"]);
        NewAddress.HouseNo = Convert.ToString(dBConnection.DataTable.Rows[Index]["HouseNo"]);
        NewAddress.Town = Convert.ToString(dBConnection.DataTable.Rows[Index]["Town"]);
        NewAddress.Street = Convert.ToString(dBConnection.DataTable.Rows[Index]["Street"]);
        NewAddress.PostCode = Convert.ToString(dBConnection.DataTable.Rows[Index]["PostCode"]);
        NewAddress.CountyCode = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["CountyCode"]);
        NewAddress.DateAdded = Convert.ToDateTime(dBConnection.DataTable.Rows[Index]["DateAdded"]);
        NewAddress.Active = Convert.ToBoolean(dBConnection.DataTable.Rows[Index]["Active"]);
        //add the blank page to the array list
        mAddressList.Add(NewAddress);
        //increase the index
        Index++;
    }
}
```

We are going to make a few minor changes to the function and then we will link the middle layer to the presentation layer.

Changing the Method to a Property

The next thing we want to do is change this function from a method to a property.

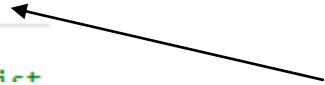
Remember the definitions of the two.

- A method allows us to perform an action on an object
- A property allows us to read or change the settings of an object

This function will allow us to access the data from the table but not do anything to it so it needs to be a property.

Modify the definition of the function by removing the brackets where the parameters would go (properties can't have parameters in C#!)

```
public void AddressList
{
    //create an array list
```



Since this is now a property not a method we need to make sure it has a getter. (This property is read only so we don't need a setter!)

Modify the function adding a getter like so...

```
public void AddressList
{
    get
    {
        //create an array list of type clsAddressPage
        List<clsAddress> mAddressList = new List<clsAddress>();
        //var to store the count of records
        Int32 RecordCount;
        //var to store the index for the loop
        Int32 Index = 0;
        //create a connection to the database
        clsDataConnection dBConnection = new clsDataConnection();
        //send a post code filter to the query
        dBConnection.AddParameter("@PostCode", "");
        //execute the query
        dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
        //get the count of records
        RecordCount = dBConnection.Count;
        //keep looping till all records are processed
        while (Index < RecordCount)
        {
            //create a blank address page
            clsAddress NewAddress = new clsAddress();
            //copy the data from the table to the RAM
            NewAddress.AddressNo = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["AddressNo"]);
            NewAddress.HouseNo = Convert.ToString(dBConnection.DataTable.Rows[Index]["HouseNo"]);
            NewAddress.Town = Convert.ToString(dBConnection.DataTable.Rows[Index]["Town"]);
            NewAddress.Street = Convert.ToString(dBConnection.DataTable.Rows[Index]["Street"]);
            NewAddress.PostCode = Convert.ToString(dBConnection.DataTable.Rows[Index]["PostCode"]);
            NewAddress.CountyCode = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["CountyCode"]);
            NewAddress.DateAdded = Convert.ToDateTime(dBConnection.DataTable.Rows[Index]["DateAdded"]);
            NewAddress.Active = Convert.ToBoolean(dBConnection.DataTable.Rows[Index]["Active"]);
            //add the blank page to the array list
            mAddressList.Add(NewAddress);
            //increase the index
            Index++;
        }
    }
}
```

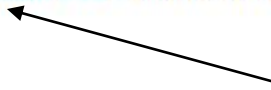
Currently the return value of the function is set to void which is no good we need the function to return some data.

Modify the function definition so that it returns a list of clsAddress...

```

public List<clsAddress> AddressList
{
    get
    {
        //create an array list of type clsAddressPage
        List<clsAddress> mAddressList = new List<clsAddress>();
        //var to store the count of records

```



As soon as you do this the “get” will be underlined in red. This is because the function must now return a list of address pages.

Add after the loop a return statement like so...

```

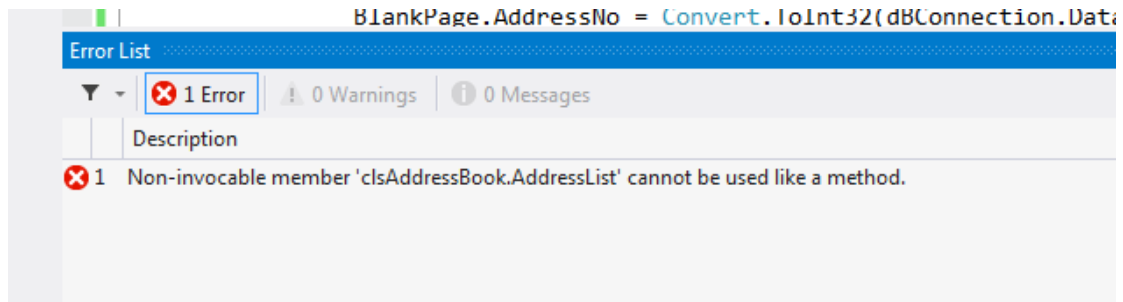
public List<clsAddress> AddressList
{
    get
    {
        //create an array list of type clsAddressPage
        List<clsAddress> mAddressList = new List<clsAddress>();
        //var to store the count of records
        Int32 RecordCount;
        //var to store the index for the loop
        Int32 Index = 0;
        //create a connection to the database
        clsDataConnection dBConnection = new clsDataConnection();
        //send a post code filter to the query
        dBConnection.AddParameter("@PostCode", "");
        //execute the query
        dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
        //get the count of records
        RecordCount = dBConnection.Count;
        //keep looping till all records are processed
        while (Index < RecordCount)
        {
            //create a blank address page
            clsAddress NewAddress = new clsAddress();
            //copy the data from the table to the RAM
            NewAddress.AddressNo = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["AddressNo"]);
            NewAddress.HouseNo = Convert.ToString(dBConnection.DataTable.Rows[Index]["HouseNo"]);
            NewAddress.Town = Convert.ToString(dBConnection.DataTable.Rows[Index]["Town"]);
            NewAddress.Street = Convert.ToString(dBConnection.DataTable.Rows[Index]["Street"]);
            NewAddress.PostCode = Convert.ToString(dBConnection.DataTable.Rows[Index]["PostCode"]);
            NewAddress.CountyCode = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["CountyCode"]);
            NewAddress.DateAdded = Convert.ToDateTime(dBConnection.DataTable.Rows[Index]["DateAdded"]);
            NewAddress.Active = Convert.ToBoolean(dBConnection.DataTable.Rows[Index]["Active"]);
            //add the blank page to the array list
            mAddressList.Add(NewAddress);
            //increase the index
            Index++;
        }
        return mAddressList;
    }
}

```



Now that the function is finished run the debugger to make sure that there is nothing wrong.

If you have done this correctly there should be an error.



Creating the Presentation Layer Code

Double click the error to see what is happening. It should take you straight to the event handler for the Display All button on Default.aspx.

Having changed the definition of the function we have to use it differently. C# won't let us treat methods like properties or vice versa.

Delete the code for the event handler so that it looks like this.

```
protected void btnDisplayAll_Click(object sender, EventArgs e)
{
}
}
```

At the bottom of the code create a new function called DisplayAddresses...

```
protected void btnDisplayAll_Click(object sender, EventArgs e)
{
}

Int32 DisplayAddresses()
{
}
}
```

Inside the function create an instance of clsAddressCollection called AddressBook.

```
Int32 DisplayAddresses()
{
    //create an instance of the address collection class
    clsAddressCollection AddressBook = new clsAddressCollection();
}
```

What we are going to do in this function is create a loop that loops through each record in the public array list adding the data to the list box.

There is a problem though.

Remember the three things we need to make a loop work.

For any loop to work we need to know three things...

1. When the loop will start
2. When the loop will end
3. Where the loop is up to at a given point

If we are to know when the loop will end we need to know the count of records in the array list.

Currently this value is hidden inside the class `clsAddressCollection`.

Due to the rules of encapsulation we cannot automatically access this value from the presentation layer.

Before we may create the loop we need to create a public `Count` property for the class.

Inside `clsAddressCollection` create the following public property...

```
public Int32 Count
    //public read only property for the count of records found
{
    get
    {
        //create a connection to the database
        clsDataConnection dBConnection = new clsDataConnection();
        //send a post code filter to the query
        dBConnection.AddParameter("@PostCode", "");
        //execute the query
        dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
        //return the count of records
        return dBConnection.Count;
    }
}
```

We may now create the presentation code here is the completed function...

```

Int32 DisplayAddresses()
{
    Int32 AddressNo;//var to store the primary key
    string Street;//var to store the street
    string PostCode; //var to store the post code
    //create an instance of the address book class
    clsAddressCollection AddressBook = new clsAddressCollection();
    Int32 RecordCount;//var to store the count of records
    Int32 Index = 0;//var to store the index for the loop
    RecordCount = AddressBook.Count;//get the count of records
    while (Index < RecordCount)//while there are records to process
    {
        AddressNo = AddressBook.AddressList[Index].AddressNo;//get the primary key
        Street = AddressBook.AddressList[Index].Street;//get the street
        PostCode = AddressBook.AddressList[Index].PostCode;//get the post code
        //create a new entry for the list box
        ListItem NewEntry = new ListItem(Street + " " + PostCode, AddressNo.ToString());
        lstAddresses.Items.Add(NewEntry);//add the address to the list
        Index++; //move the index to the next record
    }
    return RecordCount;//return the count of records found
}

```

The last step in making it work is to add a call to the function in the event handler for DisplayAll...

```

protected void btnDisplayAll_Click(object sender, EventArgs e)
{
    //display all addresses
    DisplayAddresses();
}

```

When you press the button Display All you should see the following in the list box.

Some Street
The Road
High Street

Adding Concatenation

One last tweak to the function is to extend the data displayed to the user with a bit of concatenation...

```

Int32 DisplayAddresses()
{
    Int32 AddressNo; //var to store the primary key
    string Street; //var to store the street
    string PostCode; //var to store the post code
    clsAddressBook AddressBook = new clsAddressBook(); //create an instance of the address book class
    Int32 RecordCount; //var to store the count of records
    Int32 Index = 0; //var to store the index for the loop
    RecordCount = AddressBook.Count; //get the count of records
    while (Index < RecordCount) //while there are records to process
    {
        AddressNo = AddressBook.AddressList[Index].AddressNo; //get the primary key
        Street = AddressBook.AddressList[Index].Street; //get the street
        PostCode = AddressBook.AddressList[Index].PostCode; //get the post code
        ListItem NewEntry = new ListItem(Street + " " + PostCode, AddressNo.ToString()); //create a new entry for the list box
        lstAddresses.Items.Add(NewEntry); //add the address to the list
        Index++; //move the index to the next record
    }
    return RecordCount; //return the count of records found
}

```

The ListItem Class

Notice the line of code...

```

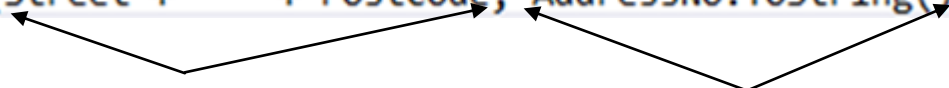
        ListItem NewEntry = new ListItem(Street + " " + PostCode, AddressNo.ToString()); //create a new entry for the list box
        lstAddresses.Items.Add(NewEntry); //add the address to the list

```

The ListItem class is a built in .NET class that allows us to prepare a single entry for the list box prior to adding it.

It accepts two parameters for its constructor...

(Street + " " + PostCode, AddressNo.ToString())



The first parameter controls what data is displayed to the user for this new entry in the list. In this case Street + " " + PostCode.

The second parameter controls how this new entry in the list is known to the system. In this case AddressNo which is the primary key of the record being added.