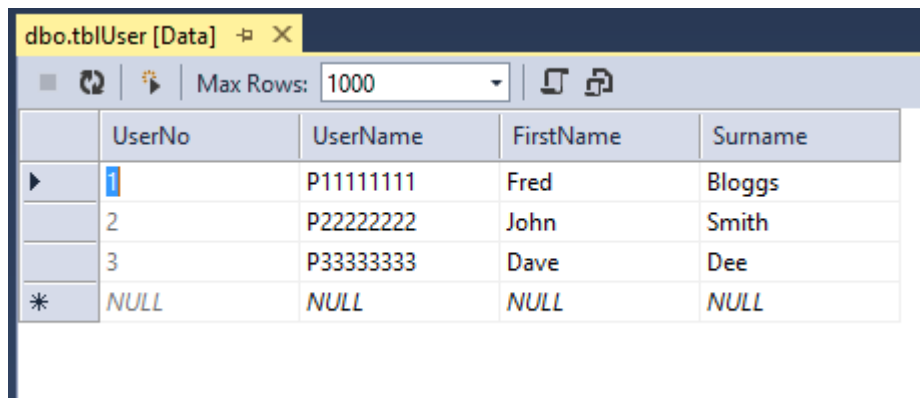


Configuring Visual Studio – Linking Presentation and Middle Layers

So far in this work we have done quite a lot of setting up Visual Studio but we haven't seen any actual data appear in any of our projects.

We have in the table tblUser the following records...



	UserNo	UserName	FirstName	Surname
▶	1	P11111111	Fred	Bloggs
	2	P22222222	John	Smith
	3	P33333333	Dave	Dee
*	NULL	NULL	NULL	NULL

It would be nice to see this on the presentation layer.

Keeping Presentation Layer Code “Thin”

When we are creating any code it is always a good idea to write with an eye to re-use. Any code in the class library may be shared across multiple projects.

Any code that is in the presentation layer is much harder to share with other projects.

.NET has a number of facilities allowing us to use as little code in the presentation layer as possible.

In IMAT1604 last year we used the following function to populate the list box on the presentation layer...

```
//function use to populate the list box
Int32 DisplayAddresses(string PostCodeFilter)
{
    ///this function accepts one parameter - the post code to filter the list on
    ///it populates the list box with data from the middle layer class
    ///it returns a single value, the number of records found
    //create a new instance of the clsAddress
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //var to store the count of records
    Int32 RecordCount;
    //var to store the house no
    string HouseNo;
    //var to store the street name
    string Street;
    //var to store the post code
    string PostCode;
    //var to store the primary key value
    string AddressNo;
```

```

//var to store the index
Int32 Index = 0;
//clear the list of any existing items
lstAddresses.Items.Clear();
//call the filter by post code method
MyAddressBook.FilterByPostCode(PostCodeFilter);
//get the count of records found
RecordCount = MyAddressBook.Count;
//loop through each record found using the index to point to each record in the data table
while (Index < RecordCount)
{
    //get the house no from the query results
    HouseNo = Convert.ToString(MyAddressBook.AddressList[Index].HouseNo);
    //get the street from the query results
    Street = Convert.ToString(MyAddressBook.AddressList[Index].Street);
    //get the post code from the query results
    PostCode = Convert.ToString(MyAddressBook.AddressList[Index].PostCode);
    //get the address no from the query results
    AddressNo = Convert.ToString(MyAddressBook.AddressList[Index].AddressNo);
    //set up a new object of class list item
    ListItem NewItem = new ListItem(HouseNo + " " + Street + " " + PostCode, AddressNo);
    //add the new item to the list
    lstAddresses.Items.Add(NewItem);
    //increment the index
    Index++;
}
//return the number of records found
return RecordCount;
}

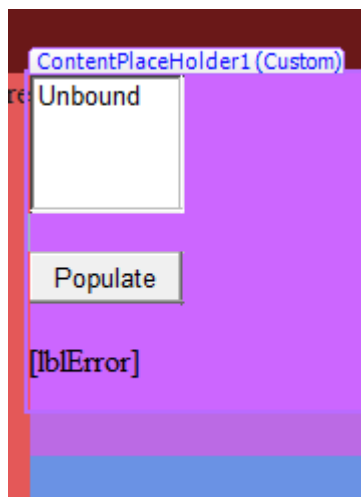
```

What if I told you that we could do the same thing with only six lines of code?

Data Binding and .NET

Right click on the project PBFrontEnd and make it the start up project.

Open Default.aspx and create a list box (lstUsers) a label control with no text (lblError) and a button (btnPopulate).



Access the event handler for the button and add the following code...

```
protected void btnPopulate_Click(object sender, EventArgs e)
{
    //call the display users function
    lblError.Text = DisplayUsers() + " found";
}
```

Now create the following function to populate the list box...

```
1 reference
Int32 DisplayUsers()
{
    //create and instance of the user collection
    clsUserCollection MyUsers = new clsUserCollection();
    //find all users
    MyUsers.FindAllUsers();
    //set the data source of the list box
    lstUsers.DataSource = MyUsers.Users;
    //set the text to be displayed
    lstUsers.DataTextField = "FirstName";
    //set the primary key
    lstUsers.DataValueField = "UserNo";
    //bind the data
    lstUsers.DataBind();
    //return the count of records in the list
    return MyUsers.Count;
}
```

Your code should look like this...

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using ClassLibrary; //add your namespace here

namespace PBFrontEnd
{
    1 reference
    public partial class Default : System.Web.UI.Page
    {
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        0 references
        protected void btnPopulate_Click(object sender, EventArgs e)
        {
            lblError.Text = DisplayUsers() + " found";
        }

        1 reference
        Int32 DisplayUsers()
        {
            //create and instance of the user collection
            clsUserCollection MyUsers = new clsUserCollection();
            //find all users
            MyUsers.FindAllUsers();
            //set the data source of the list box
            lstUsers.DataSource = MyUsers.Users;
            //set the text to be displayed
            lstUsers.DataTextField = "FirstName";
            //set the primary key
            lstUsers.DataValueField = "UserNo";
            //bind the data
            lstUsers.DataBind();
            //return the count of records in the list
            return MyUsers.Count;
        }
    }
}

```

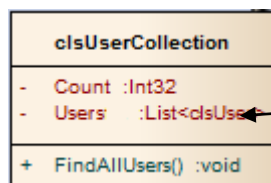
Don't forget to add your namespace to the list of files this code is "using" at the top!

Pressing the Populate button should produce something like the following...



So what is going on?

If we look at the design for the class `clsUserCollection` we see the following...



The users are presented as a handy list via the public `Users` property.

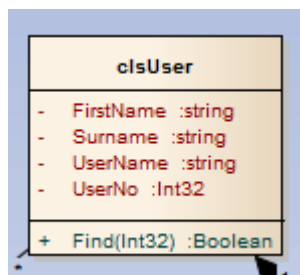
.NET list boxes are smart.

The first thing we tell the list is where there is a list that contains data it can use.

```
//set the data source of the list box  
lstUsers.DataSource = MyUsers.Users;
```

Next we need to tell the list box which properties to use.

The list of `Users` is based on `clsUser` which has the following structure.



The following code tells the list firstly which property to display and then which property contains the primary key.

```
//set the text to be displayed
lstUsers.DataTextField = "FirstName";
//set the primary key value
lstUsers.DataValueField = "UserNo";
```

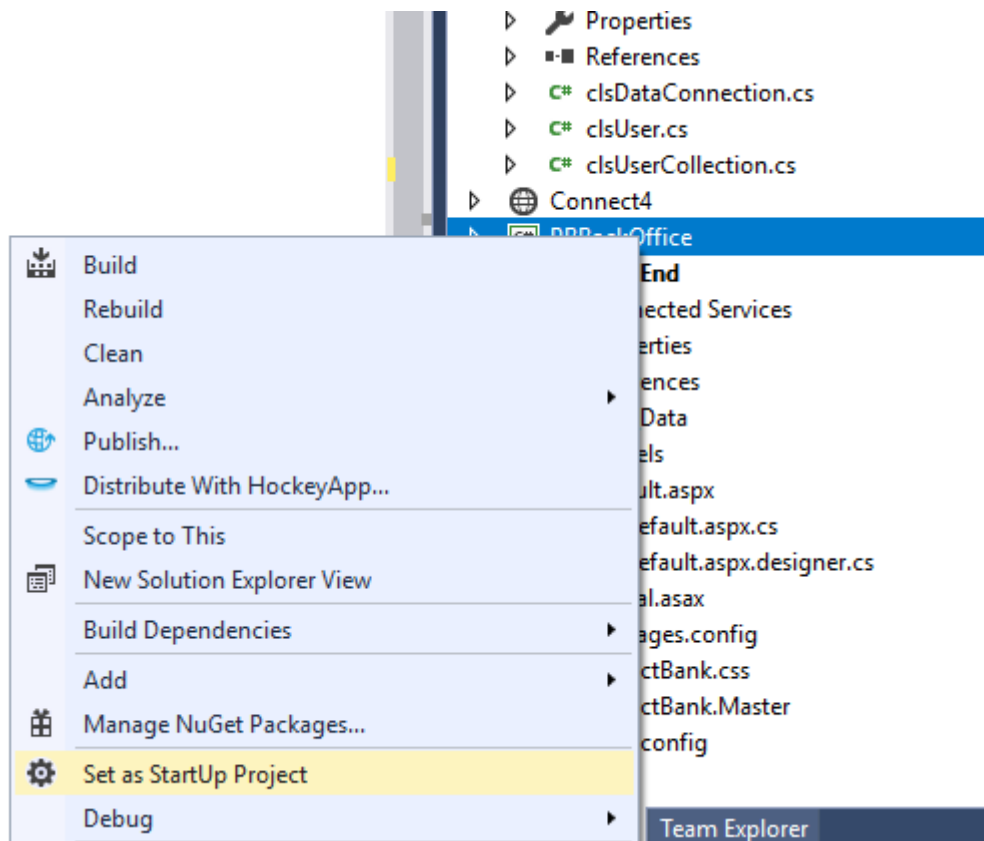
The final step is to bind the list box to the list in the collection class.

```
//bind the data
lstUsers.DataBind();
```

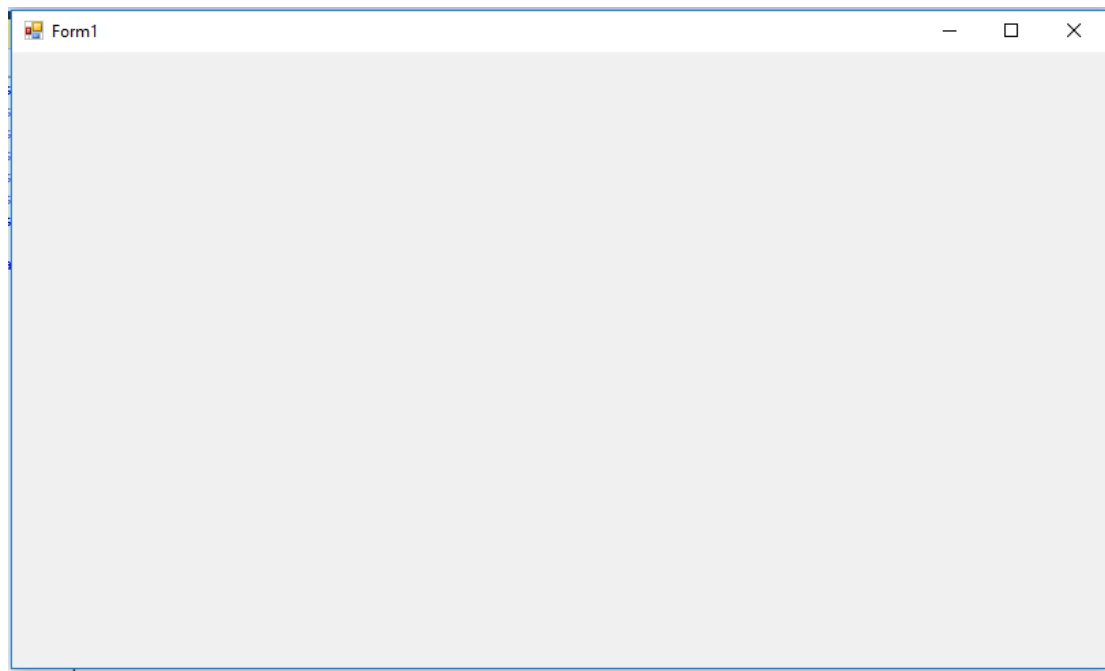
Data Binding on the Project Bank Back End

Having displayed some data in a list on the front end we will do the same in the back office part of the system.

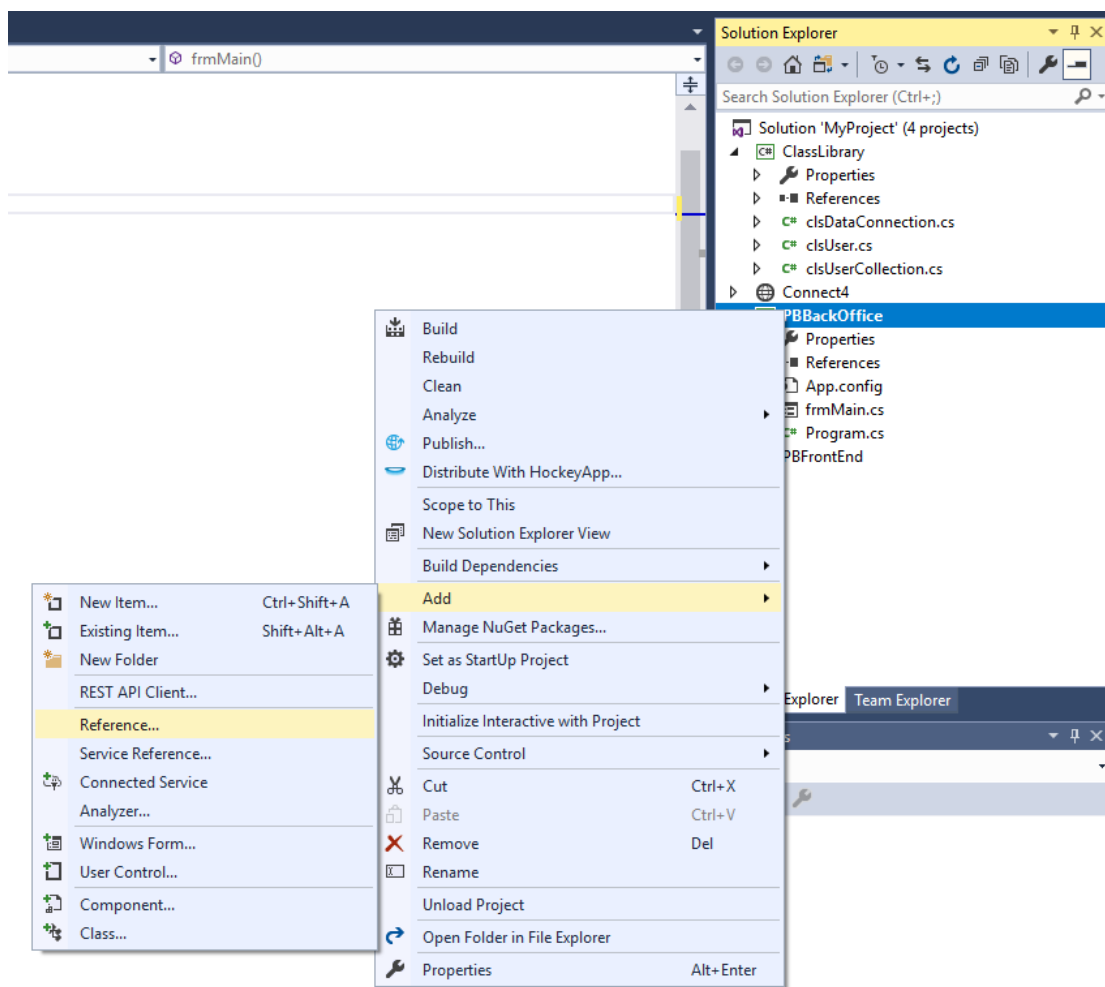
Right click on PBBackOffice and make it the start-up project...



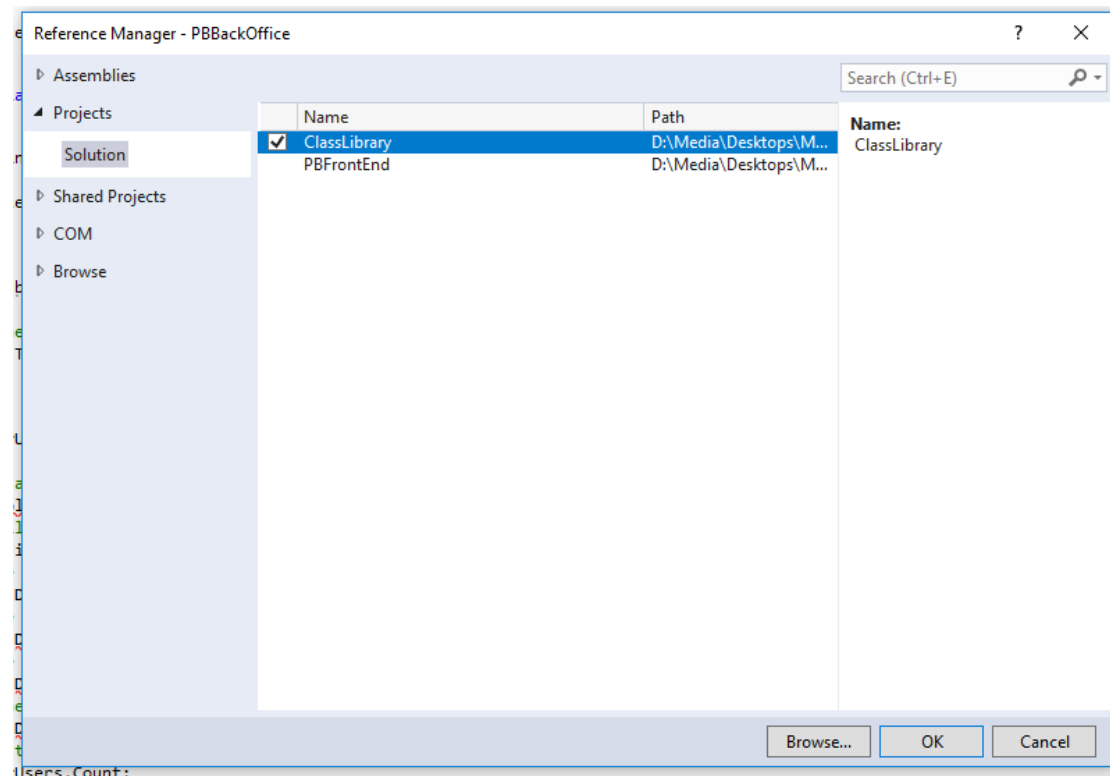
Run the program just to remind you what it does...



Before doing anything else we need to tell the project about the class library so that it is able to use the code there...



Followed by...

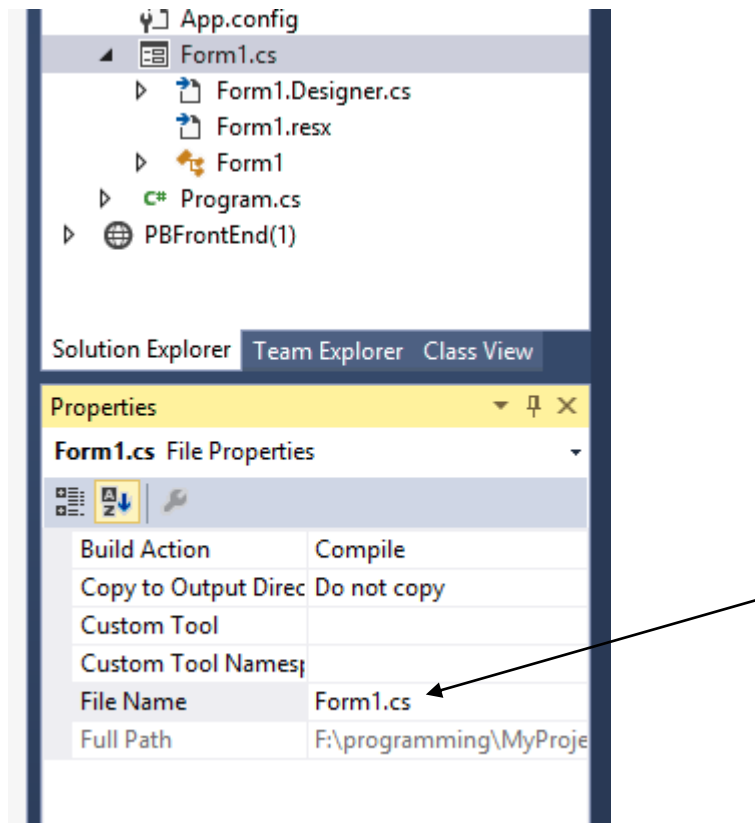


Rather than being a web site this project is a Windows Desktop application.

Set up the form Form1.cs like so...

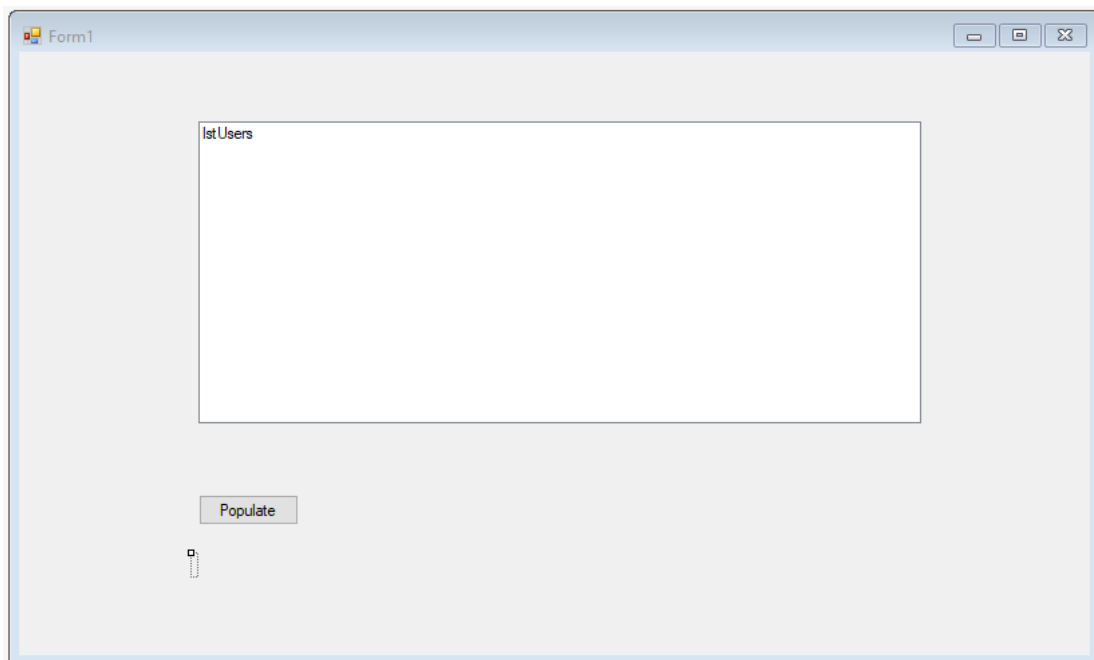
First change the name of the file to something more meaningful.

Click once on Form1.cs and locate the file name property...



Change this to frmMain.cs

Use the same properties as you did for the web front end and set it up something like so...



At the top of the code for the form you will need to add a using to make the form access the class library...

tice

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using ClassLibrary; ←
```

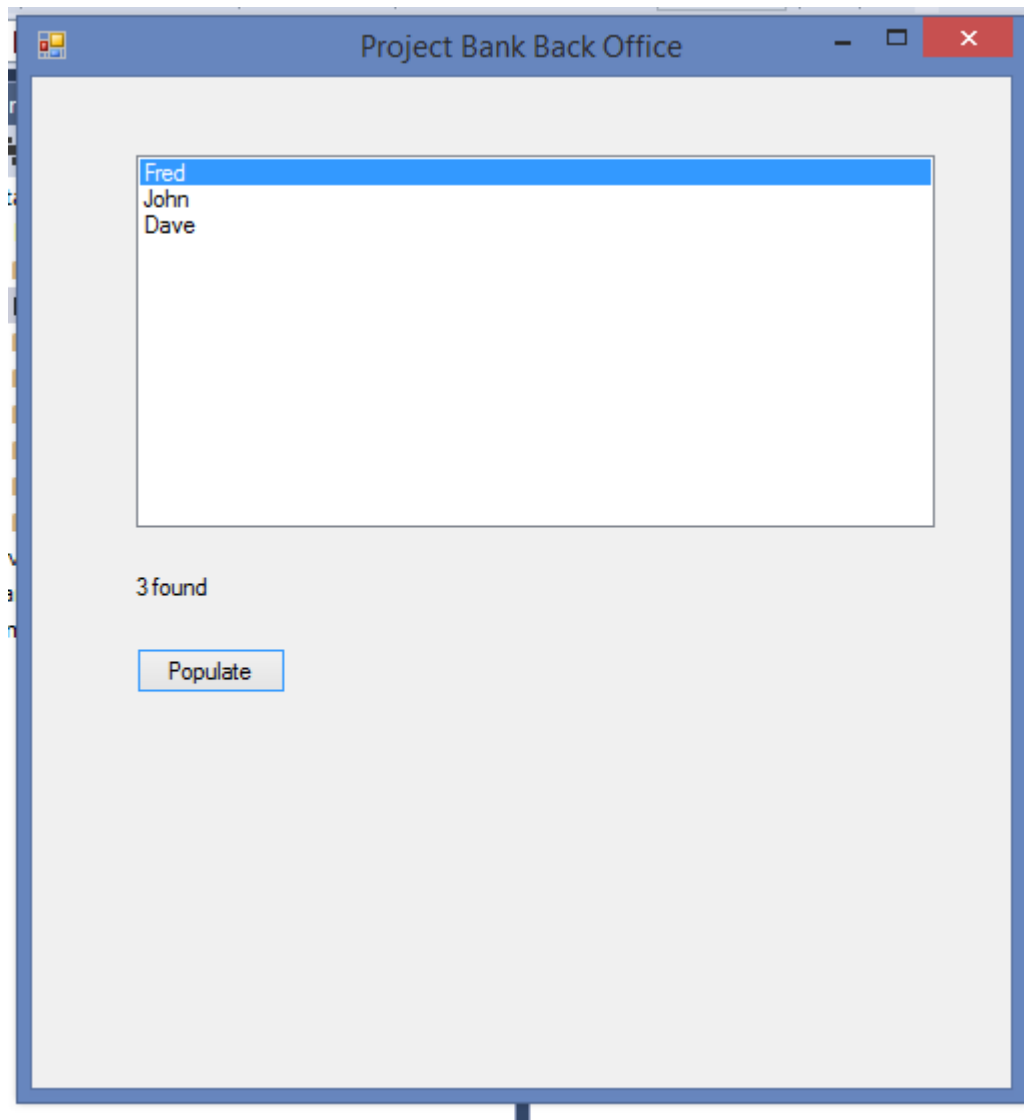
Access the event handler for the populate button and add the following code.

```
private void btnPopulate_Click(object sender, EventArgs e)
{
    //call the display users function
    lblError.Text = DisplayUsers() + " found";
}
```

Now create the function DisplayUsers...

```
1 reference
Int32 DisplayUsers()
{
    //create and instance of the user collection
    clsUserCollection MyUsers = new clsUserCollection();
    //find all users
    MyUsers.FindAllUsers();
    //set the data source of the list box
    lstUsers.DataSource = MyUsers.Users;
    //set the text to be displayed
    lstUsers.DisplayMember = "FirstName";
    //set the primary key
    lstUsers.ValueMember = "UserNo";
    //return the count of records in the list
    return MyUsers.Count;
}
```

Run the program and press the button. You should see something like the following...



The important thing to note here is that the web front end and the windows back end not only share the same data they also share the same code.

Any changes we make to the class library will automatically be passed on to both applications (and any other projects sharing the same library)

Improving the Back End Design

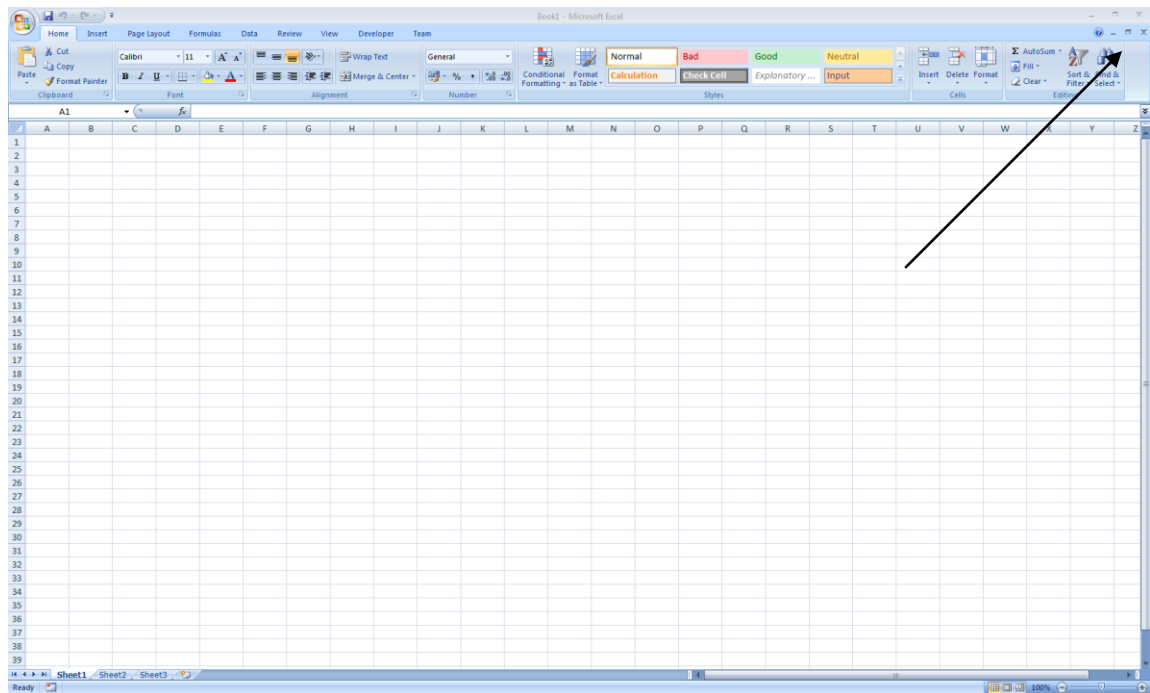
One issue that you will encounter as you develop the back end of the system is that the more windows forms you create the more messy the design will become.

To manage the forms appropriately we will set up what is called a Multiple Document Interface (MDI) application.

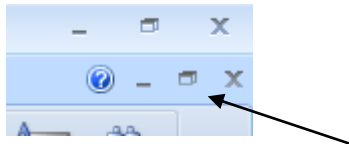
Introduction to MDI Applications

If you have used MS Excel you will already have seen an MDI application.

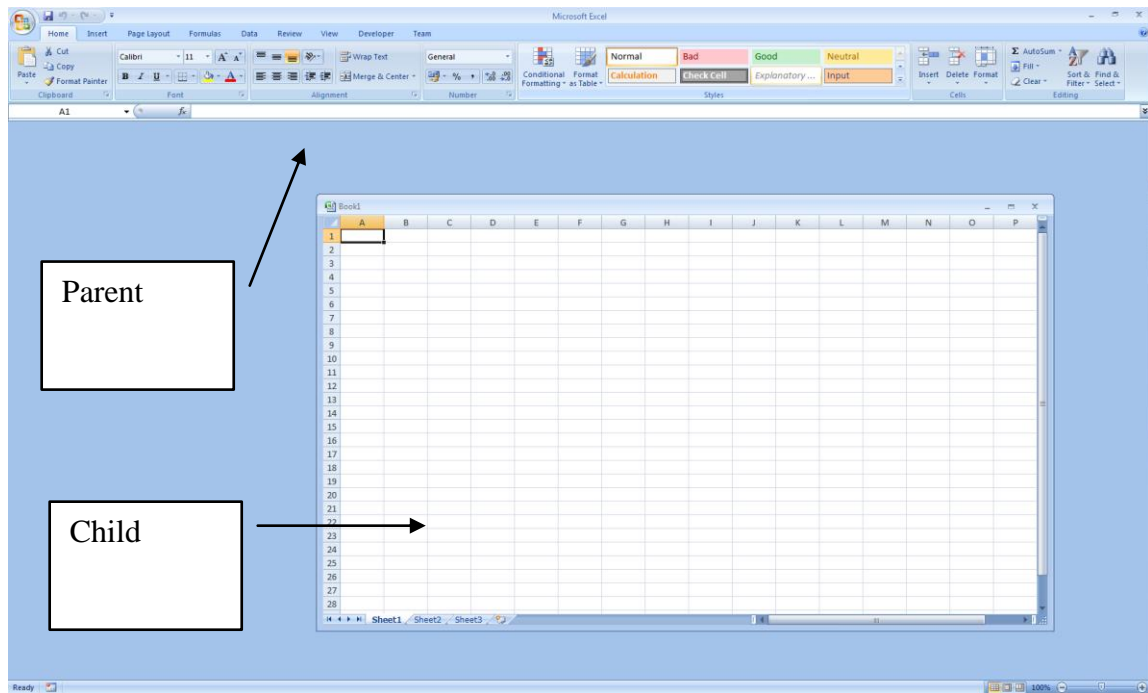
Below is a screen shot of Excel...



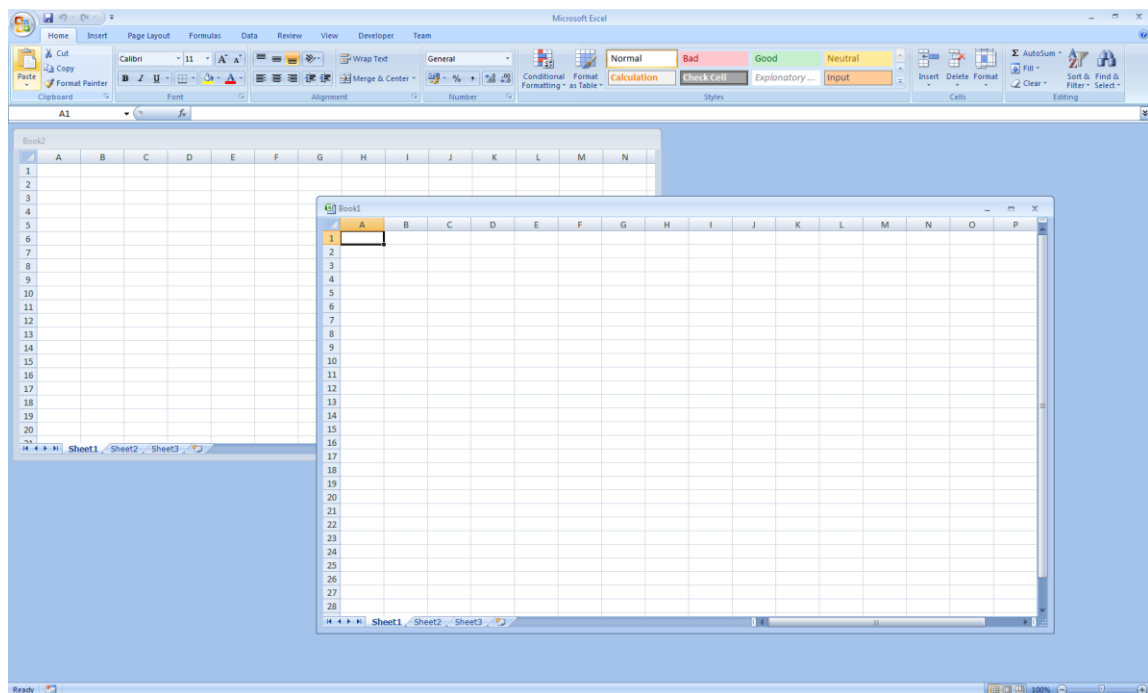
Notice in the top right hand corner of the application there are two sets of max min and close controls...



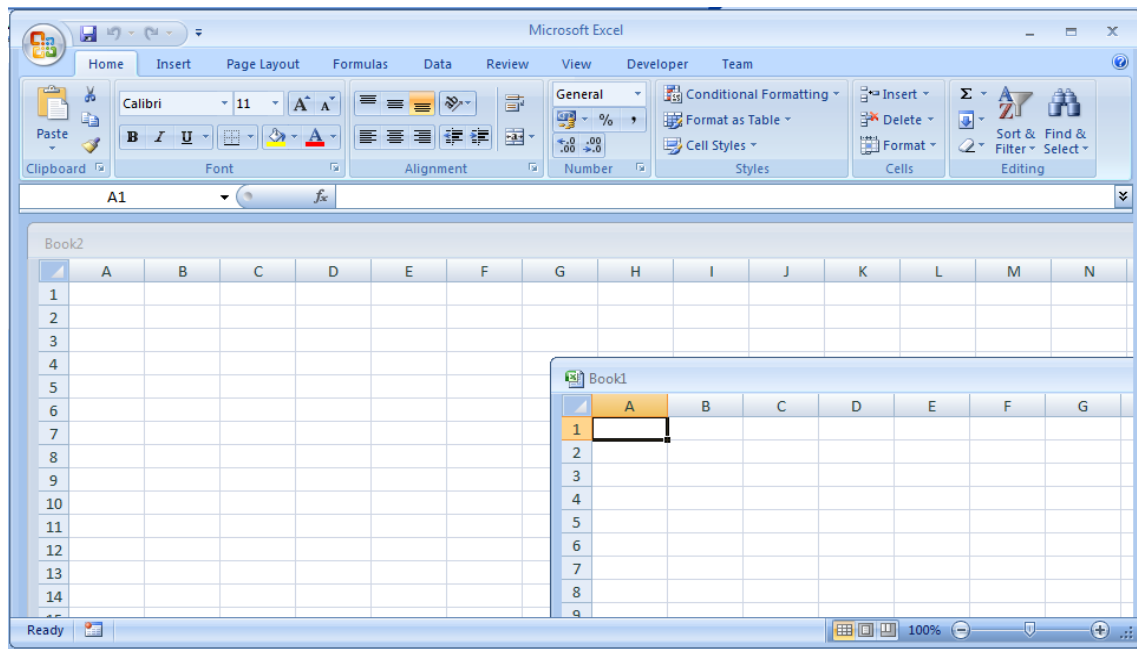
If you reduce the size of the child window by clicking on the icon above you will see the following happen...



If we create a second spreadsheet we see a second child document.



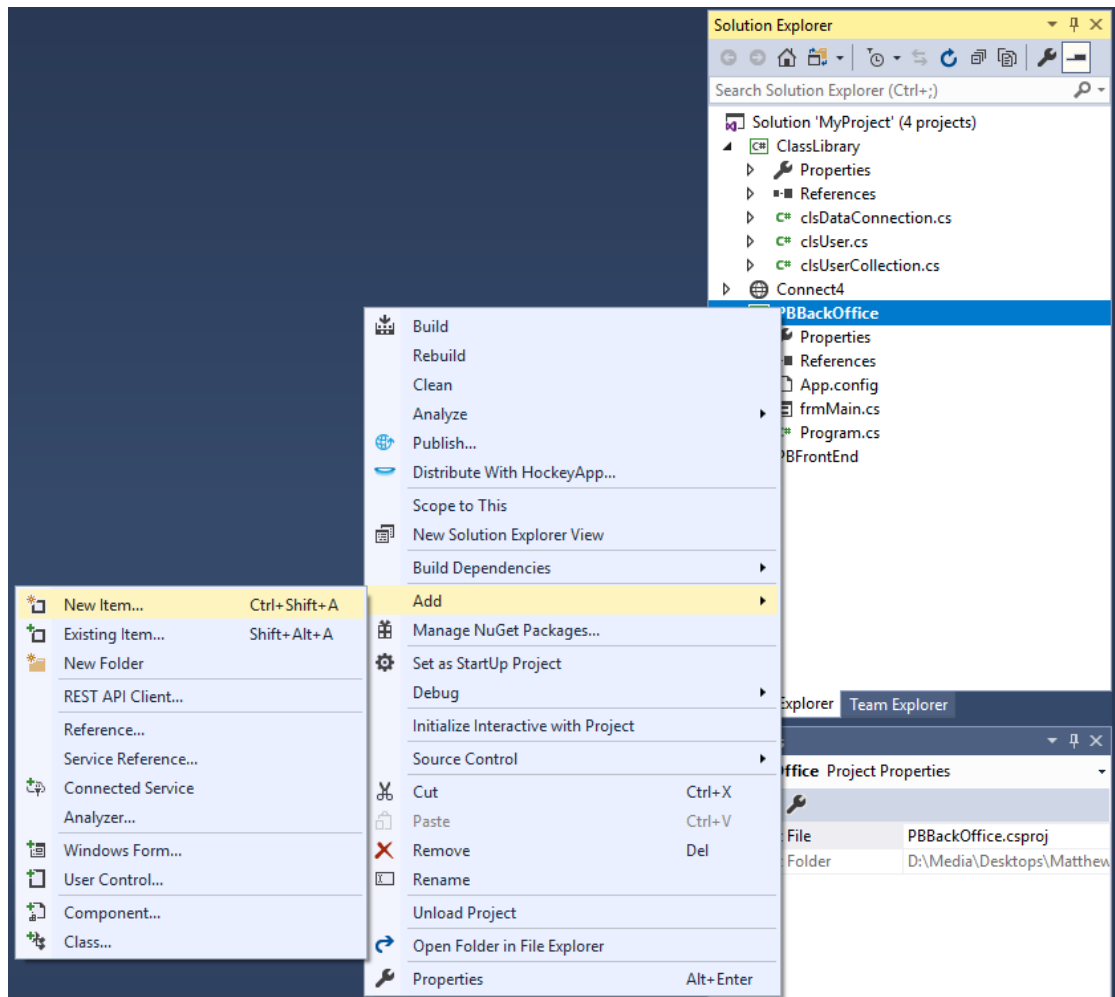
Also note that both child documents are managed by the parent document such that if you reduce the size of the parent the child forms become partially hidden.



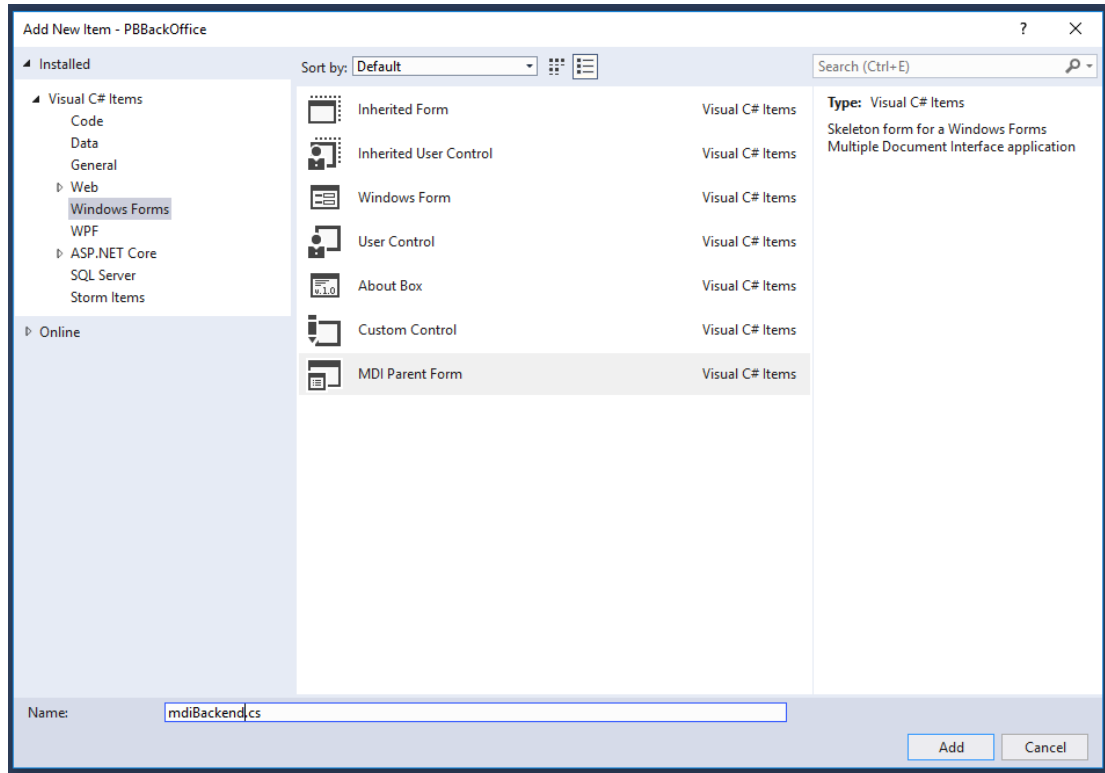
Creating an MDI Application

Having already created the back end project all we need to do is add an MDI form to the project and make it the parent for any child forms.

Add a new item to the back office project...

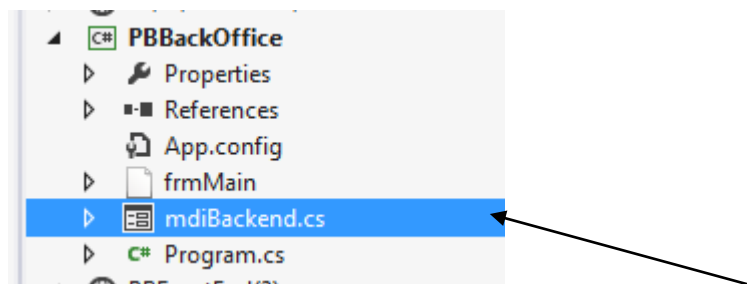


Under Windows Forms, locate MDI Parent Form...

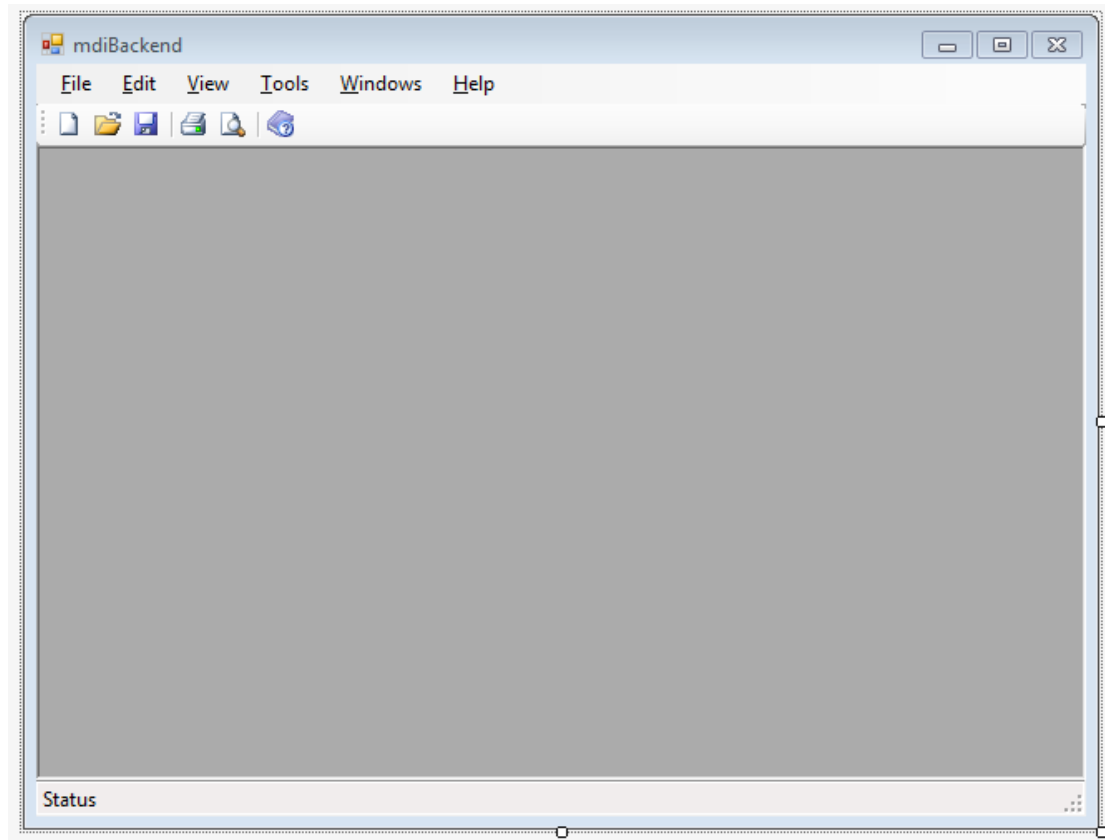


Set the name of the form to mdiBackend.cs

This will add the MDI parent to the solution explorer like so.

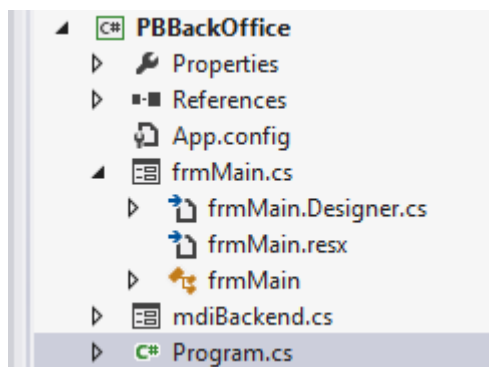


You will also see the MDI parent form displayed.



Now that the MDI parent form is present we need to set it as the start up form for this project.

Open the file Program.cs in the solution explorer...



Change the code as follows to make the MDI for the startup object..

```

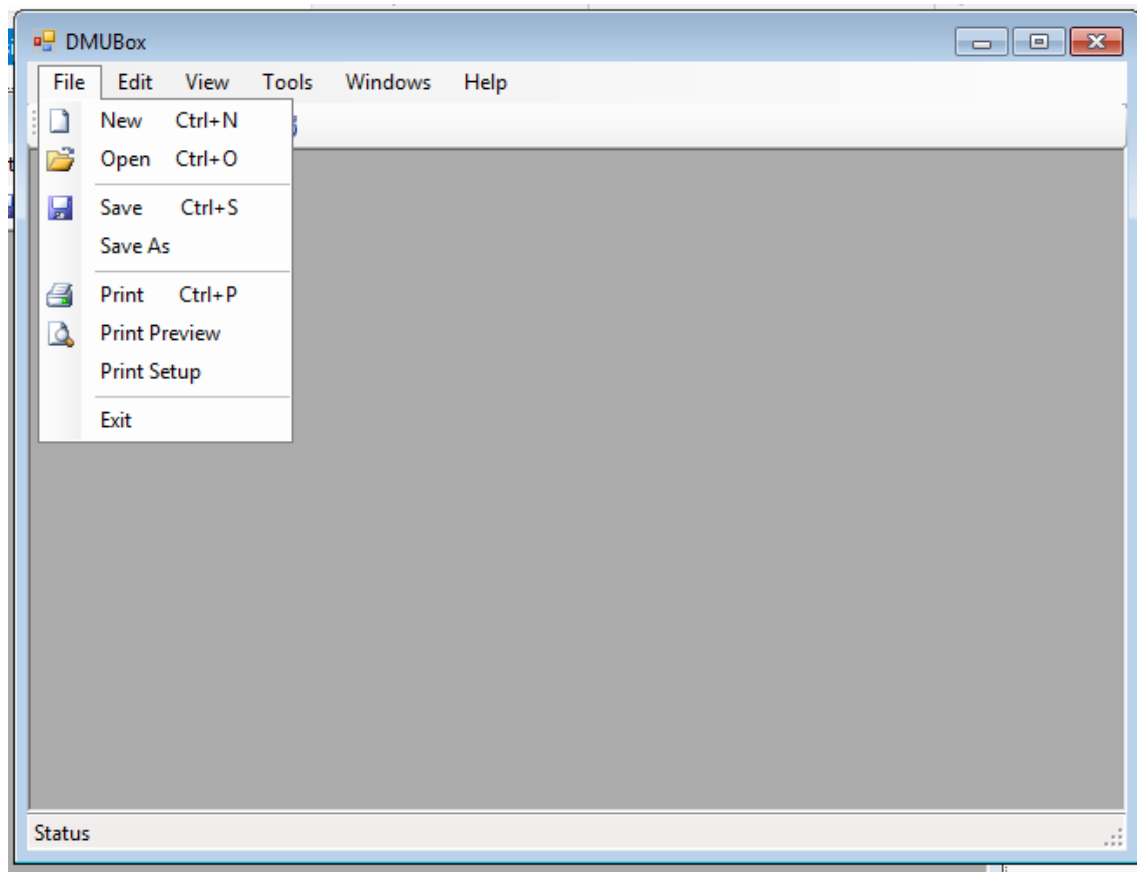
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PBBackOffice
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new mdiBackend());
        }
    }
}

```

Now that the application is configured correctly you may run it by pressing F5.

You should see the following...

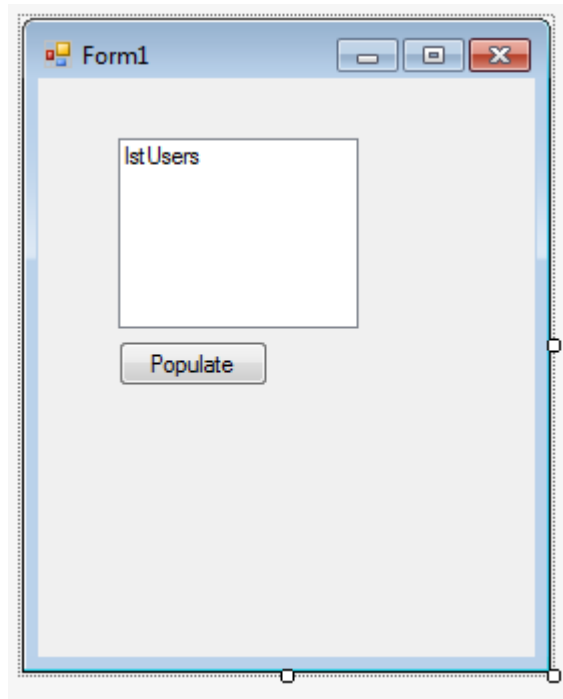


Examine the menu options and note how many things have been automatically for you. We probably won't need all of these options however having so much generated for you makes coding much simpler.

Configuring Child Forms

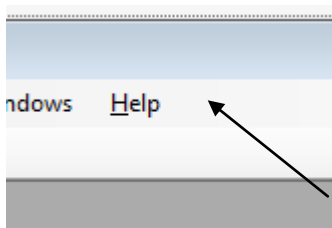
Now that the application is set with the MDI form as the main form we need to set about using child forms.

We should already have a child form `frmMain.cs`



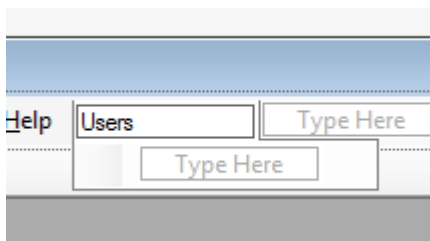
Creating Menu Options

On the main MDI form click once on the menu bar to the right of the word Help.

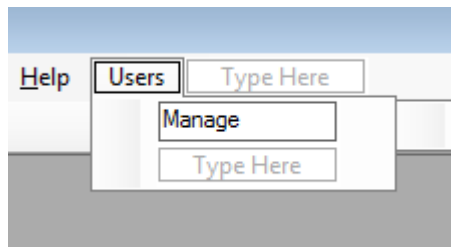


A space should appear allowing you to create your own menu heading.

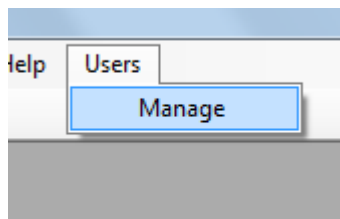
Enter the text Users...



Below this enter the name Manage...



Run the program to see the new menu option working...



Adding Keyboard Shortcuts

Modify the text in the new menu options to &Users and &Manage.

Notice what happens if you run the program and press ALT and U on the keyboard.

Adding the Code to Display the Form

To access the event handler double click the word Boxes.

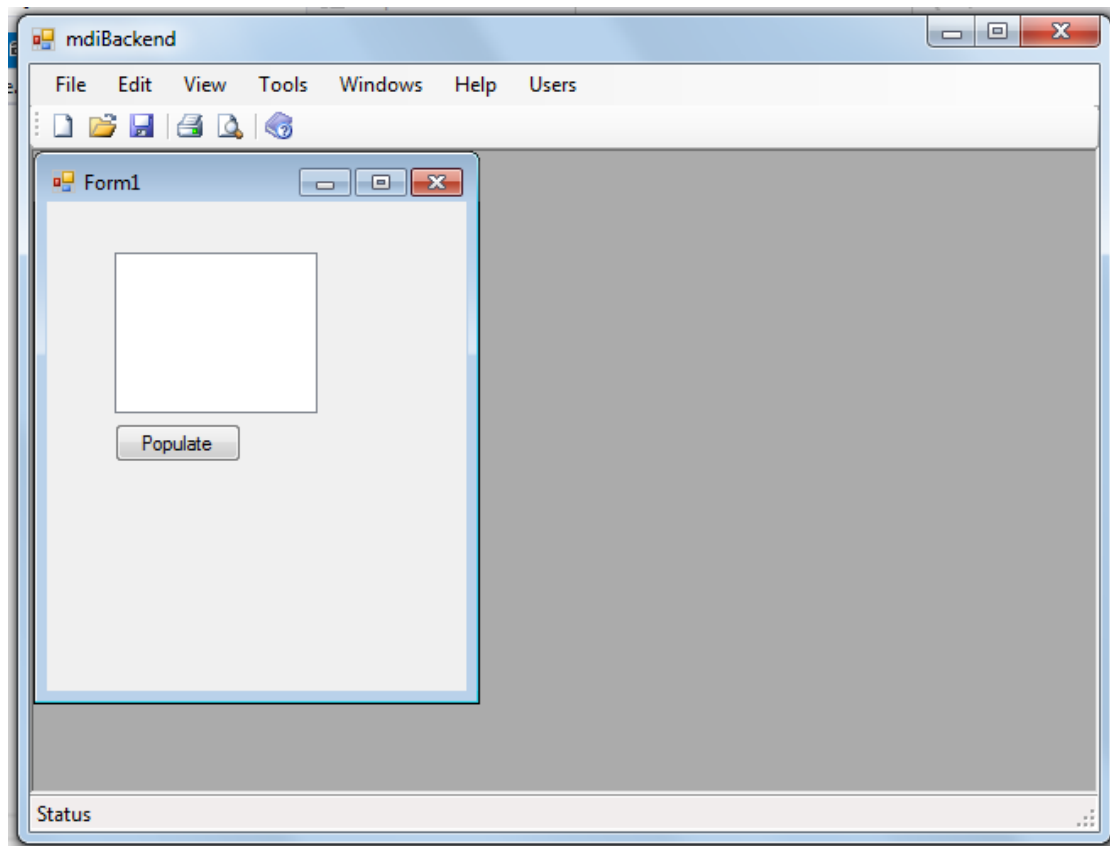
This will display the event handler stub.

```
private void manageToolStripMenuItem_Click(object sender, EventArgs e)
{
    |
}
```

Modify the event handler using the following code...

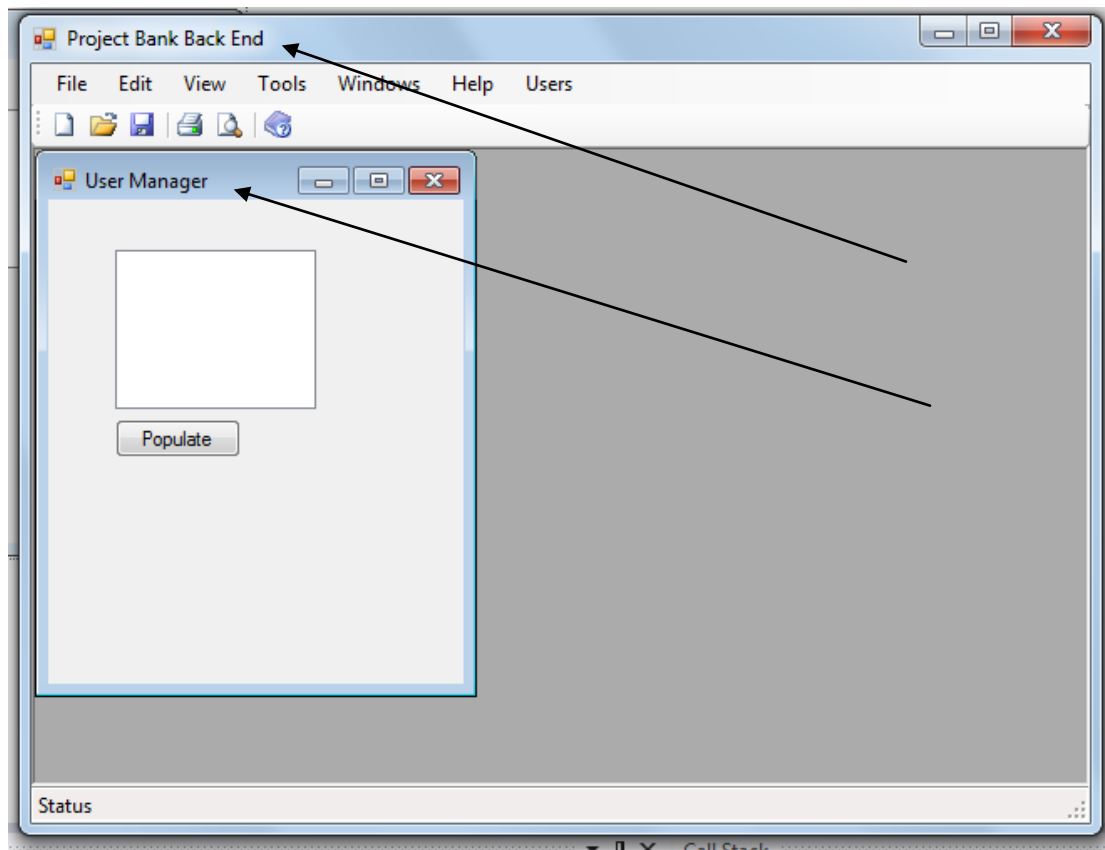
```
private void manageToolStripMenuItem_Click(object sender, EventArgs e)
{
    //create an object based on frmMain
    frmMain userList = new frmMain();
    //make the object a child of the mdi parent
    userList.MdiParent = this;
    //make the form visible
    userList.Visible = true;
}
```

Now test your program to see what happens when you select your menu option. What you should see is an instance of your child form displayed within the MDI parent...



Tidy up the interface by getting rid of unnecessary menu options and also set the title of your forms.

For example...



Not

