

C# Tutorial

This tutorial provides an introduction to coding for mobile devices using C# and Visual Studio. There are many topics to cover, so unfortunately we will not be able to go into much depth in the time available. However, feel free to contact any of the organisers after the tutorial if you have any questions. We are all postgraduate students who completed our undergraduate degrees in computing science at Glasgow, and we are happy to chat about your projects or anything else to do with software development.

Marek Bell, Malcolm Hall & Julie Maitland

Office: F141

marek@dcs.gla.ac.uk, mh@dcs.gla.ac.uk, jules@dcs.gla.ac.uk

Topics covered

This is a basic introduction to coding C# for mobile devices. However, at the end of the tutorial you should be able to:-

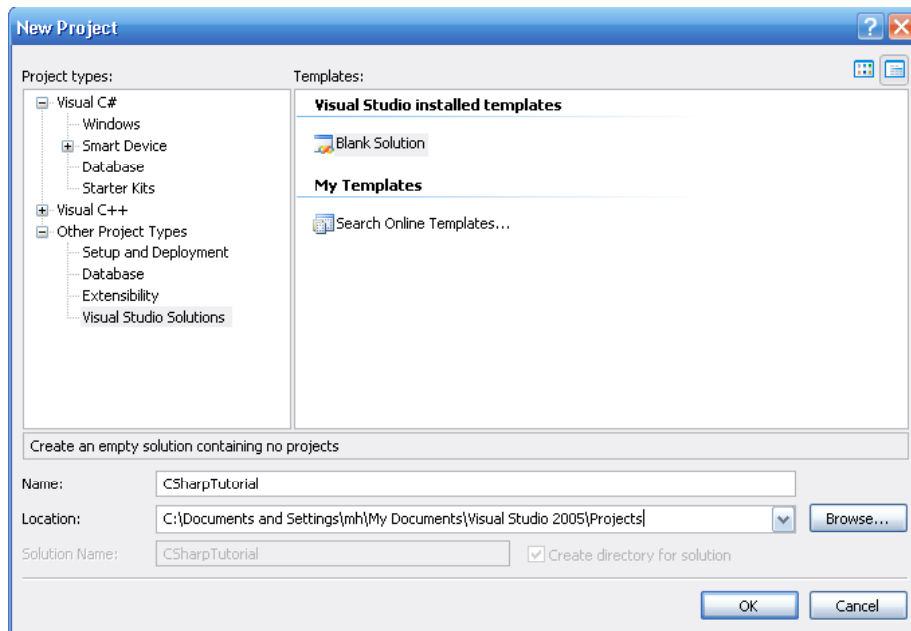
- Create a basic GUI application for Windows desktop machines
- Create a basic GUI application for Windows Mobile devices
- Create your own Dynamic Link Libraries (DLLs)
- Link to and use DLLs in your code
- Call native Windows API code from inside C# applications

It is vital that throughout the tutorial you try to understand the code you are entering and what it does. Please ask about anything that might not be clear in the tutorial text or anything you want more information about.

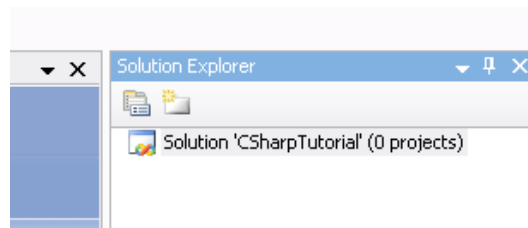
Creating a solution

Before we begin coding, we must create a solution in Visual Studio 2005. A solution can be thought of as a meta-project. It can store many sub-projects within it, and is simply a useful way of organising your development tasks.

Open Visual Studio 2005 (Start->Programs->Compilers) and select File->New->Project from the menu. A window will appear showing different project types. Expand "Other Project Types" and choose "Visual Studio Solutions". Name the solution 'CSharpTutorial' and place it somewhere in the 'My Documents' folder. By default it will show My Documents\Visual Studio 2005\Projects, which is fine. Click 'OK' to continue.



The solution you created will appear in the 'Solution Explorer' in the top right window.

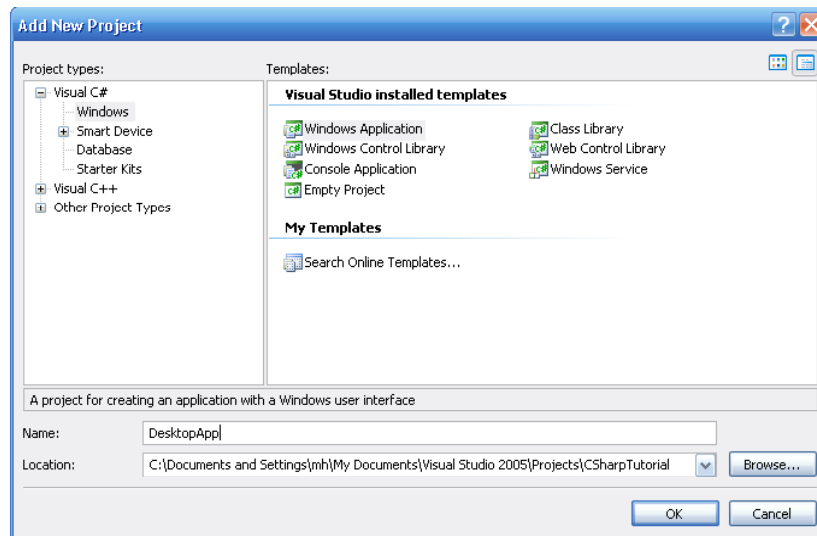


To simplify things, we will store all the code we write today within this one solution.

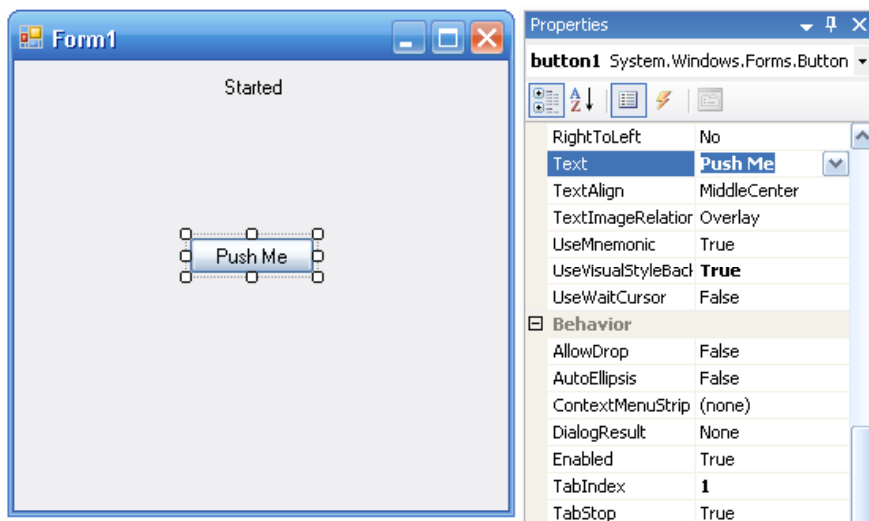
Windows GUI Application

We will start with the easiest task – creating a simple application that will run on the Windows desktop. Our first application will just display a message when a button is clicked.

Right-click the CSharpTutorial entry in the Solution Explorer at the top-right of Visual Studio and select Add->New Project from the pop-up menu. Again you will be shown the project type chooser, this time select **'Visual C# Projects'** (careful not to chose Visual Basic) in the Project Types area and choose 'Windows Application' from templates. Name the project 'DesktopApp' and then click OK. The default project location is the solution folder which is perfect.



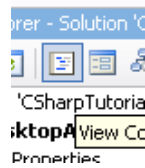
The new project will be created within your solution and 3 files are created in the new project. Program.cs which contains the “static void main” method which is responsible for starting the application, Form1.cs which is where you will enter the code for the application, and if you expand Form1.cs you will see Form1.Designer.cs which is where the auto-generated code by the designer will appear. You should never edit the designer code manually because visual studio may overwrite your changes. Open and look at all of these files if you like, some of the code is hidden using a feature called outlining, click the plus to expand the code, or right click in the code and choose “stop outlining”. You can switch between design view and code view of Form1 by right clicking in the window. Switch to design view and form the toolbox on the left, drag a label and a button onto the form. Drag the items around to lay them out as shown below. Select the label and in the properties window at the bottom left change its ‘Text’ entry from ‘label1’ to ‘Started’. Notice the label is resized to fit the longer text, this is due to the AutoResize property being true. Select the button and change its text to ‘Push Me’.



The GUI components of the form are now complete and all we have to do is add the code that changes the label text when the button is clicked. In the designer, double-click the “Push Me” button. When you do this Visual Studio creates a method which

20-10-2006

hooks into the button event and takes you to that new method in the code view. You can switch between GUI view and code view using these buttons:

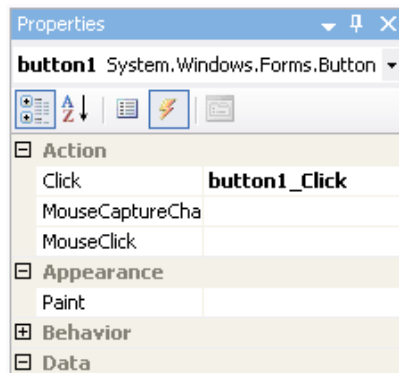


In the code view simply add the following code into the button1_Click method...

```
label1.Text = "Hello World!";
```

Notice when you press the fullstop a pop up is displayed with all of the possible methods and properties available for label1. You can display this auto-complete pop up at any time by pressing ctrl-space. This is the best way to enter error free code quickly, and means you can work quickly without searching API documentation.

Double clicking the button to create a click event is a special. If you would like to add event handlers for other events on GUI controls then switch to the designer and click the lightning bolt. The events for that control are listed and event handlers can be added by double clicking the field to the right of the event, and again it will create the method stub and take you to the code. We won't add any other ones just now though.



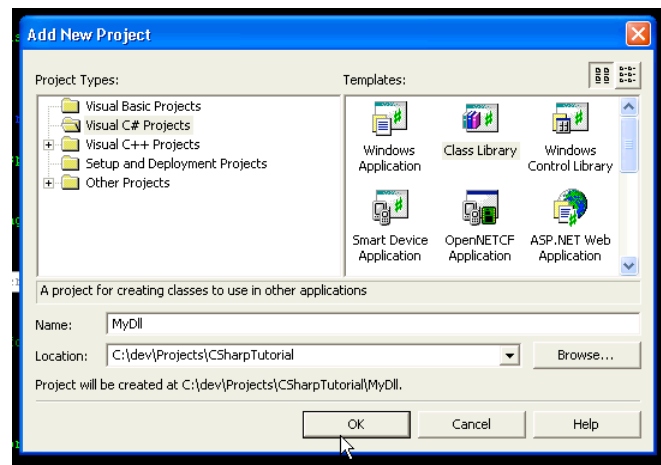
The application is now complete. You can compile and run it in one step by pushing Ctrl-F5 on the keyboard (this is the shortcut for the menu Debug->Start Without Debugging). When you click the button the text should change, also notice the label resizes to fit the new text, the AutoResize property works at runtime too.

What you have created is a fully working Windows application (if you navigate through Explorer you can see the actual executable file in the "bin" directory of the project folder and run it from there too).

DLL (Class Library)

Next we will create and use our own Dynamic Link Library. DLLs are vital as they allow developers to package, distribute and reuse code in different applications or different versions of the same program.

Right-click the CSharpTutorial solution in the Solution Explorer and select Add New Project from the menu. This time, pick Class Library in the template area and enter "MyDll" as the name before clicking OK.



As most DLLs do not contain any GUI components unless you create them yourself, Visual Studio takes you straight to the code view. We will just write one simple method so change the entire contents to...

```
using System;

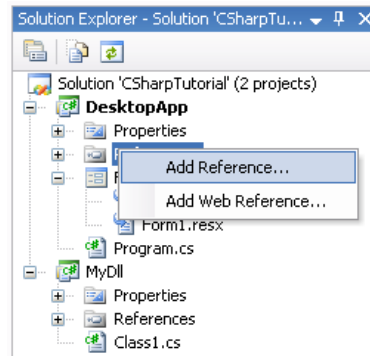
namespace MyDll
{
    public class Utils
    {
        public static int FindMax(int[] haystack)
        {
            int max = int.MinValue; // this is the lowest possible int value
            foreach (int i in haystack){
                if(i > max)
                    max = i;
            }
            return max;
        }
    }
}
```

We can also rename Class1.cs in the project to Utils.cs however this isn't a necessity. Note the foreach loop which may be new to you if you have not used C# before. Basically, it's just a short way of writing for (int i = 0; i < haystack.length; i++). Using foreach also means that instead of writing haystack[i] inside the loop you can just write i on its own instead.

Check that the code compiles by building the solution. To do this press Ctrl-Shift-B (this is the shortcut for the menu Build->Build Solution).

20-10-2006

Once your DLL is building we can use it in our GUI app. To achieve this you have to tell the GUI app about the new DLL by creating a reference to it. Under DesktopApp in the Solution Explorer right-click references and select Add Reference.



References can be added to existing DLL files, common DLLs or our own projects. As this DLL is our own project, select the Projects tab, click MyDll, click select and then click OK.

Now we can change the GUI code to use the method in the DLL. Double-click Form1 and then go to the code-view. The first change to make is to mark in the file that we are going to be using MyDll. Scroll to the top of the code and beside all the other 'using' statements add: (remember you can use the ctrl+space to autocomplete – this also lets us verify the dll has been referenced)

```
using MyDll;
```

Now, go back to the button1_Click code and change it to read... (our previous line has been commented out using //)

```
private void button1_Click(object sender, System.EventArgs e)
{
    //label1.Text = "Hello World!";
    int[] searchArray = {5, 85, 34, 96, 1, 942, 3, 48};
    int highest = Utils.FindMax(searchArray);
    label1.Text = Highest number is " + highest;
}
```

Compile and run the program again using Ctrl-F5. If you look in the bin/Debug folder for the application you should see that there is now an additional MyDll.dll file. Just like any other Windows app that uses a DLL, when the executable runs it looks for the DLL it requires and links to it dynamically at runtime (hence the name, Dynamic Link Library). Note that the executable file and the DLL file are independent. This means that you could replace the code in the DLL to a more efficient version or even to do something completely different and your existing executable would use the new code without you having to recompile it. This is constantly occurring when you update Windows or any other programs – the DLLs are being updated but the main program executables are not.

Using Native code

C#, and all .NET languages, compile to bytecode and run in the CLR (Common Language Runtime). This is similar to the way in which Java compiles to bytecode and runs on the JVM (Java Virtual Machine). Often, however, you will want to use native code – either for maximum speed or to access lower-level things. One extremely common example of this is using any of the Windows API methods as these all exist in native DLLs in the Windows\System32 directory. Luckily, it is extremely easy to call into native code from C# (Compared with the Java Native Interface). We will demonstrate by using a Windows API call to get the name of the currently logged in user.

First, we have to tell our code that we want to do some interoperation between .NET code and native code at runtime. For our example we will also be using some basic text services as well so scroll back up to the using statements and add both of these...

```
using System.Runtime.InteropServices;
```

Next, we have to tell our class what native method we want to call and in which DLL it can be found. On a new line inside the class (anywhere after public class Form1) add the code...

```
[DllImport("advapi32.dll")]
public static extern bool GetUserName(StringBuilder buffer, ref int size);
```

This DllImport statement tells .NET that we are importing the GetUserName method from the “advapi32” DLL. We can now call the method just like we would call any C# method. Return to the button1_Click method for one last time and change it to read...

```
private void button1_Click(object sender, System.EventArgs e)
{
    StringBuilder sb = new StringBuilder(100); // a StringBuilder is just a dynamic size string
    int length = sb.Capacity;
    GetUserName(sb, ref length);
    label1.Text = "Hello " + sb;
}
```

The equivalent C++ method for the GetUserName function looks like...
 BOOL GetUserName(LPWSTR lpBuffer, LPDWORD pcbBuffer)
 ...so StringBuilder is smart enough to convert itself to a LPWSTR when calling into a native library.

Compile and run the application. When you click the button the program dynamically links to the DLL it finds in Windows\System32\ and uses its native code to find the method entry point using the supplied method name.

Note this isn't a particularly useful p/invoke, since Environment.UserName will give us the same thing. However if we want to use Bluetooth or perform scans for WiFi access points then we will need to p/invoke since .NET doesn't include these things, and once you know which .NET types map to C++ ones its pretty straight forward.

Pocket PC application

This final section of the tutorial guides you through creating a Pocket PC application that links to a mapping dll created by us and we have used it in many of our research systems. It has features such as displaying map layers and markers, and you can pan and zoom the view. This is again an example of how useful DLLs can be; you easily have access to a powerful mapping kit without having to read or understand any of the mapping source code. Pocket PC applications are called “smart device applications” and they use the .NET compact framework. The compact framework is a subset of the full .NET framework which we were using for the Windows application, so it takes up less memory on constrained smart devices. Usually you can work around the missing methods by writing some extra code yourself.

Currently there are 2 versions of the compact framework available. CF2 (released with Visual Studio 2005) is much more powerful, however it does need to be installed to the device separately which is often a source of confusion for pocket pc users, and they complain about the size of it (its 4mb and most devices have 64Mb of storage), consequently there isn't much software available which uses it. CF1 is built into every modern Microsoft Pocket PC and Smartphone so if you are writing software for the general public sometimes it can give a better user experience if you put in some extra effort to make it run on the more primitive CF1, which is what we are going to use now.

Add another project to the solution. This time expand “Smart Device” and then Pocket PC 2003, and choose Device Application(1.0) as the template and name the project “PdaApp” before clicking OK.

The device we are developing for today is the HP iPAQ hx2750 and runs Pocket PC 2003. Newer devices run Windows Mobile 5 and if the SDK is installed you can target those devices too, email support if you would like to do that. The only real difference is a change to the designer view, and a couple of extra WM5 specific libraries. We prefer 2003 devices for our research work since there was a change to how files are stored in WM5 (now in the flash ROM instead of the RAM) and they are actually quite a bit slower to develop with.

The smart device GUI designer is slightly different from the Windows designer in that the menu bar is at the bottom and there is an image of a device. Now we are going to make 2 menu items on the menu bar. Just click on the left area of the bar and type in “Exit”, then to the right of that make another item “Zoom”. You can rename these items to `exitMenuItem` and `zoomMenuItem` to make your code more readable, use the Name field in properties.



The reason we require an exit menu item this time is because Pocket PC applications do not properly exit when you click the “X”. Applications that run on the desktop fully close when you click the “X” but on PDAs clicking the “X” just makes the application minimize, we can’t deploy and run new versions of our application if the old executable one is still in use. Therefore, we need another menu item that does cause the app to fully close. Double-click the exit button to add the click event handler and in the method in code-view just add the following simple code for closing the app...

```
Close();
```

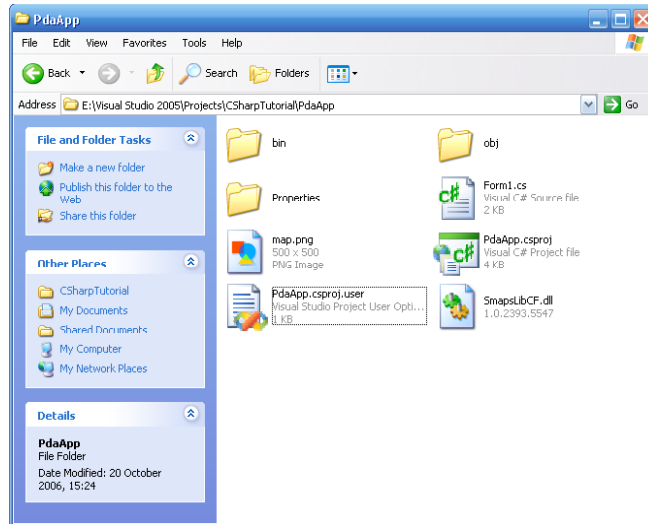
Before we add in the map let’s test the exit. First, right-click the PdaApp project in the solution explorer and select “Set as StartUp Project” from the menu. All this does is tell Visual Studio that you want this application, rather than the previous desktop application, to start when you hit Ctrl-F5. Notice it changes to bold. Go ahead and hit Ctrl-F5 now to run the program. A Pocket PC 2003 emulator will start up and your program will be deployed to and run on it. When the app starts up just click the exit button to make sure it closes properly. You can then close the emulator but it won’t do any harm if you leave it open. If you do close it then make sure you select “Save emulator state” so it starts up quicker next time.

Now we can add the map in. Download the DLL and the map image from...

<http://www.dcs.gla.ac.uk/~hall/hci4/maps.zip>

Extract the contents of the zip and place the two files in the CSharpTutorial\PdaApp folder.

20-10-2006



Go back to Visual Studio and right-click the references in PdaApp to add a reference to the new DLL. This time, select Browse in the first window. It should open in the correct folder so just select the new SmapsLibCF.dll file. Now add the map images by right-clicking the actual PdaApp itself and selecting “Add->Add Existing Item”. Change the file type to images and then select the map.png. Now select map.png in the project and change the “Copy to output dir” property to “Copy if newer”.

Back in the code-view for the form go to the top and add the following using statement...

```
using SmapsLibCF;
```

We only need one extra global variable – a Viewer which is used to display the actual map. Somewhere in the class itself, probably best beside the other variables above the constructor, add...

```
private Viewer viewer;
```

In the form constructor, after InitializeComponent() method call (which is the method which runs the designer generated code) insert the following...

```
viewer = new Viewer();  
viewer.useTemplate = false;  
GeoRectangle rect = new GeoRectangle(0, 0, 100, 100); // we could of used real latitudes and longitudes  
ImageLayer mapLayer = new ImageLayer(@"Program Files\PdaApp\map.png", rect);  
viewer.AddLayer(mapLayer);  
viewer.Width = this.Width;  
viewer.Height = this.Height;  
this.Controls.Add(viewer);
```

This creates the Viewer, adds a map layer to it using one of the gif images and finally makes the viewer fill the window.

Go ahead and run the app to test it. A map should appear and you should be able to pan around it by dragging the mouse.

The final task is just to make the zooming button work. Back in the GUI designer double-click the zoom menu item to hook into the event handler method. See if you

20-10-2006

can fill in the code for the method yourself so that when you click the button it activates the zoom and if you click the button again it returns to panning. The viewer behaviour can be set using...

```
viewer.SetTool("toolname");
```

...where the tool name is either "ZoomTool" or "PanTool". The menu item text can be set in exactly the same way you set the label text in both the desktop hello world and the using native code examples.

Deploy and run your code again (Ctrl+F5) and see if it works. To zoom the map you drag up and down, dragging left and right in zoom mode has no effect.



To test this on the desktop change this line to:
`ImageLayer mapLayer = new ImageLayer("map.gif", rect);`

And copy the images into the bin folder where the exe exists.

Deploying to the device

Call over a demonstrator and they will show you how to deploy to a real device.

Resources

The .NET Framework community. Forums, technical articles, samples
<http://samples.gotdotnet.com/quickstart/>

MSDN Library .NET Framework reference guide
http://msdn.microsoft.com/library/en-us/netstart/html/cpframeworkref_start.asp

20-10-2006

OpenNetCF is an open source project that adds more features to the compact framework, often matching the same APIs found in the full framework. Since CF2 was released it has become a bit redundant, but they still have some useful classes and a good forum.

<http://www.opennetcf.org>

XDA-Developers is a forum and wiki with lots of information about HTC manufactured windows mobile PDAs and phones, including unlocking and modifying them. If you have seen an Orange, Vodafone or O2 Windows Mobile device it is bound to be an HTC device, as they just rebrand them.

<http://forum.xda-developers.com/>