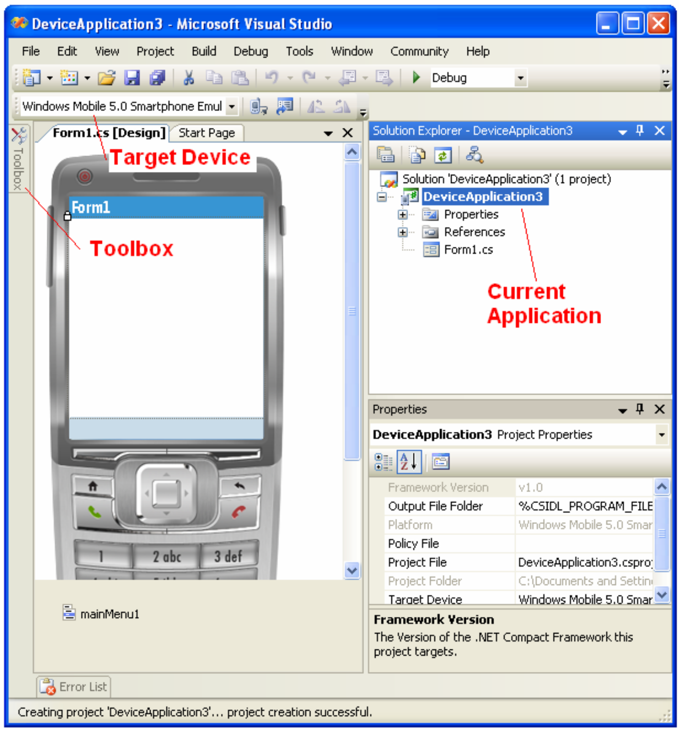
Smartphone Development Tutorial

CS 160, March 7, 2006

Creating a simple application in Visual Studio 2005 and running it using the emulator


DeviceApplication3 - Microsoft Visual Studio
File Edit View Project Build Debug Tools Window Community Help
Debug
Windows Mobile 5.0 Smartphone Emul
Toolbox
Form1.cs [Design]
Start Page
Target Device
Form1
Toolbox
mainMenu1
Solution Explorer - DeviceApplication3
Solution 'DeviceApplication3' (1 project)
DeviceApplication3
Properties
References
Form1.cs
Current Application
Error List

Properties

DeviceApplication3 Project Properties

Framework Version	v1.0
Output File Folder	%CSIDL_PROGRAM_FILE
Platform	Windows Mobile 5.0 Smar
Policy File	
Project File	DeviceApplication3.cspro
Project Folder	C:\Documents and Settin
Target Device	Windows Mobile 5.0 Smar

Framework Version
The Version of the .NET Compact Framework this project targets.

Creating project 'DeviceApplication3'... project creation successful.

1. In Visual Studio 2005, create a project for the Smartphone platform (Smart Device→Windows Mobile 5.0 Smartphone). To make deploying to the actual phone work, you should choose the "Device Application (1.0)" template. (The phone doesn't have enough memory to install version 2.0 of the compact framework.)
2. Set the "Target device" to "Windows Mobile 5 Smartphone QVGA Emulator".
    - Right-clicking your current application in the Solution Explorer, selecting "Properties", and selecting the "Devices" tab. (The application is the second item on the list, not the top level 'Solution' entry.)
    - Change the target device using the drop-down menu.
    - Also uncheck the "Deploy the latest version of the .NET Compact Framework" checkbox.
3. By default, the "Device Application (1.0)" template includes a main form for your application, called *Form1*. Double-clicking "Form1.cs" in the Solution Explorer will open the form for editing.
    - Select the form by clicking on the picture of the phone in the main editor window
    - In the "Properties" panel at the bottom-right corner of the screen, find *FormFactor* and change the value to "Windows Mobile 5.0 Smartphone QVGA." This will make the display of the phone the same size as the actual Smartphones used in class.
4. Drag some controls from the Toolbox (by default, hidden at the left side of the screen) onto the phone in the main editor window.
5. Under the "Debug" menu, select "Start Debugging." This will bring up a dialog allowing you to select a deploy target – Pick "Windows Mobile 5.0 Smartphone Emulator QVGA" and click deploy. This will launch the emulator and launch your application. (Selecting "Debug" → "Stop Debugging" will quit your application but will leave the emulator running.)

This process works fine for running applications from Visual Studio. But it doesn't allow you to browse the file system on the emulator, which is necessary if you want to run the program manually or to manipulate files that the application creates on the phone. For that, you need to use ActiveSync.

1. Under “Tools” select “Device Emulator Manager” which opens the Manager. Scroll down the list of Emulated devices until you see “Windows Mobile 5.0 Smartphone QVGA Emulator” and select it. Now under the “Actions” menu in the Emulator manager, select “Cradle”.
2. Now open ActiveSync. Under “File” → “Connection Settings” check “Allow connections to one of the following” and select “DMA”. You only need to do this once. When you click “Connect”, ActiveSync should connect to the Emulator and give you some setup screens. After the first time, the connection should happen automatically when you “cradle” the Emulator via the Device Emulator manager.
3. Now you can browse the file system on the Emulator by selecting “Explore” on ActiveSync (or selecting “Mobile Device” from My Computer). By default, applications are deployed to “Program Files”.
4. To launch your application from the Start Menu of the Emulated device, select the executable (in “Program Files\[app name]”), and right click and select “Create Shortcut”. Right click on the shortcut and do “Cut”. Then navigate the Emulator File system to “\Windows\Start Menu\Debug Apps” and right click and do “Paste”. Now go to the Emulator. Open its start menu and scroll down to “Debug apps” and open it. You should find the app link in there, and selecting it will run your program.

Running the application on a Smartphone

Like most commercial Smart Phones, the SP5s have strong security protection by default. All the code that runs on the phone must be signed. The Smartphone SDK includes test signature certificates and programs. Unfortunately, the default security on the phone is too high to run even these. The first step is therefore to change the local security policy. You need a (signed) version of the Registry Editor to do this.

1. **Configuring the Phone’s Security Policy**
   WARNING: This requires a registry edit. We suggest this be done by a member of your group who has used a Windows registry editor before. If you’re unsure about this step, please bring the phone to us and we will make the edits.
   a. Copy the file regeditSTG.zip (http://inst.eecs.berkeley.edu/~cs160/sp06/readings/regeditSTG.zip) to your PC, and unzip it to produce regeditSTG.exe. Connect your Smartphone to the PC, and pair it with ActiveSync. Then click on “Explore” in ActiveSync to open the Smartphone’s file system. Browse to the \Program Files directory, and create a new folder called “Regedit”. Copy (drag and drop) the file regeditSTG.exe into this directory.
   b. You should then be able to browse to this directory from the phone itself. Click the start button on the phone, go down one screen to “File Manager”, and browse to \Program Files\Regedit. Click to start the registry editor. Once it starts, scroll to:
   HKEY_LOCAL_MACHINE\Security\Policies\Policies
   and then click on “Values”. You should see a list of Numerical key value names, starting with 00001001.
   Select 00001001 and change the “Value data” field to 1 (decimal). Click on Done.

   Exit the registry editor (red hangup button). This change allows the remote API (RAPI) tool to make further security changes.

   c. Now make sure that your device is connected to the PC with ActiveSync, and go to the Windows Mobile SDK directory on your PC (C:\Program Files\Windows CE Tools\wce500\Windows Mobile 5.0 Smartphone SDK\Tools) and run: *RapiConfig.exe /p SdkCerts.xml* which makes various configuration changes to allow execution of Visual Studio apps on the phone.
   d. At this point you will be able to compile code from Visual Studio and deploy and run it on the Device. If the code is unsigned, you will get a security prompt the first time you run each program. To avoid this, you can configure Visual Studio to sign your application.
2. **Configuring Visual Studio to sign your app**
   a. In Visual Studio set "Target device" to "Windows Mobile 5.0 Smartphone Device".
   b. From the "Project" menu, select "Properties", then "Devices". Select "Sign the project output with this certificate". Then click on "Select Certificate→Manage Certificats→Import" to open the certificate import wizard. The certificate to use is located in the Windows Mobile SDK, and its default location is C:\Program Files\Windows CE Tools\wce500\Windows Mobile 5.0 Smartphone SDK\Tools\SDKSamplePrivDeveloper.pfx (copy and paste this into the "file name" field). During certificate import, you may be prompted for a password, but you can leave it blank. Assuming this works, Visual Studio will sign your app each time you build it. You should do this for every project you plan to run on the phone.
3. **Building and Running**
   a. From Visual Studio, make sure "Target Device" is set to "Windows Mobile 5.0 Smartphone Device", and build the app as with the Emulator. Use "Build→Deploy [app name]" to transfer it to the phone. It should appear in a directory \Program Files\[app name] on the Smartphone. You can run it from there using File Manager from the phone, or you can create a shortcut as you did with the Emulator to be able to run it directly from the phone's start menu. You can also run (and Debug) it as before, by selecting "Start Debugging" from the "Debug" menu.

Making a more complicated application

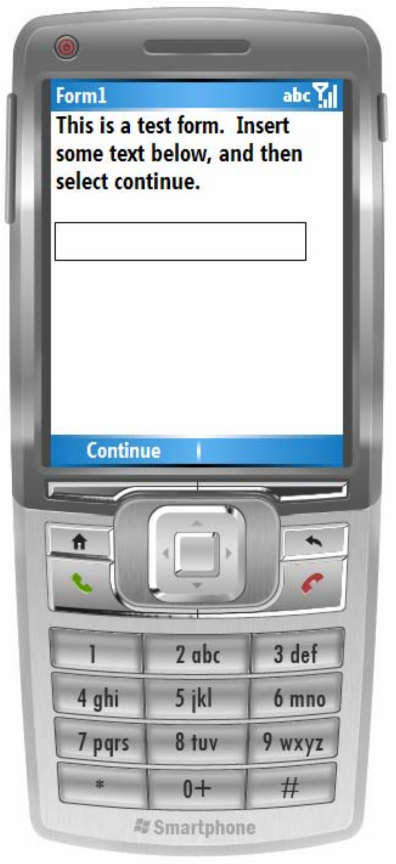
The following will walk you through a slightly more complicated example making use of multiple forms and dialog boxes.

1. Add a Label to *Form1* with the text "This is a test form. Insert some text below, and then select Continue." (Change the text by editing the *Text* property in the property editor.)
2. Add a TextBox to *Form1*, remove the default text (by changing the *Text* property), and change the *Name* property of the TextBox to "m_textBox".
3. Click on the menu area of *Form1* (the bottom of the display, right above the left menu button). A "Type Here" prompt should appear, allowing you to type in "Continue". This creates a new MenuItem and automatically places it in the main menu of *Form1*.

4. You should now have a display that looks something like the sample shown here. However, the application doesn't do anything yet…
5. Visual Studio will help you automatically connect event handlers to user interface elements. To connect one to the "Continue" menu item, simply double-click on it. Doing so will generate the function *menuItem1_Click* in Form1.cs and will attach the appropriate event handler to the control. (The event handling code is in the "Windows Form Designer generated code" section of Form1.cs, which is hidden by default.)
    a. Event handling code: `this.menuItem1.Click += new System.EventHandler(this.menuItem1_Click);`
    b. Callback function: `private void menuItem1_Click(object sender, EventArgs e) { }`
6. Now, whenever the user selects the "Continue" menu item, the function *menuItem1_Click* will be called automatically.
7. Insert the following code into the body of the *menuItem1_Click* function:

```
private void menuItem1_Click(object sender,
EventArgs e)
{
      // Show a dialog asking the user if it's okay
      to continue
      DialogResult result = MessageBox.Show("You
        entered: " +
        m_textBox.Text + ". Do you want to continue?",
        "Continue?", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question,
        MessageBoxDefaultButton.Button1);

      // Open up form2 if it is okay
      if (result == DialogResult.Yes)
      {
          Form2 nextForm = new Form2();
          nextForm.Show();
       }
}
```



This is a test form. Insert some text below, and then select continue.

This code will show a dialog box asking the user if it's okay to continue. If the user selects "Yes", it will open up a new instance of *Form2*.

8. Now create a new Form class called *Form2* (right-click on your application in the Solution Explorer, select "Add", and select "Windows Form…".
    a. Put anything you want in the body of the form
    b. Create a menu item with the text "Exit."
    c. Attach a new event handler to the "Exit" menu item, and enter `Application.Exit();` into the body of the function (which will cause your application to quit when it executes).

Using the Experience Sampling toolkit

If you have time, try out the experience sampling toolkit, which allows you to instrument your application with automated logging of program state, user response time, and responses to custom questions.

- Sample code is available at http://inst.eecs.berkeley.edu/~cs160/sp06/section/ExperienceLogger.zip
  - Open the solution in Visual Studio and look at ESMTestApplication for basic functionality.
- Compiled code is available at http://inst.eecs.berkeley.edu/~cs160/sp06/section/ExperienceLoggerDLLs.zip
  - Link to ExperienceLoggerCF1.0.dll in your application by right-clicking on “References” in the Solution Explorer, selecting “Add reference…” and browsing to find the downloaded Dll.

Using OpenNETCF

In building your applications, you may also want to use the OpentNETCF.org *Smart Device Framework (v 1.4)*, which includes C# managed code accessors to advanced phone functionality (e.g. vibrating, SMS, etc.).

- More information at http://www.opennetcf.org/
- Binary download at http://www.opennetcf.org/getfile.asp?file=SDF14Assemblies&dir=bin
- Documentation at http://www.opennetcf.org/library/