Programming for mobiles

theory;
different OS;
frameworks;
good practices.

prof. O. Nakov, Ph.D

Accessing devices in mobile programming

1. Device audio and video playback capabilities

1.1 Audio

Audio capture code must reside into individual classes within your container.
Each audio interface is particular to the platform.

Android SDK

Access to the device microphone is using Android.Media.MediaRecoreder object, witch instance supports set of properties and methods to configure and manage the audio stream.

Windows Phone 7

Access to the device microphone is using Microsoft.Xna.Framework.Audio.Microfone object.

Playing audio

Android

MediaManager object plays sounds and video from variety of sources. You can use Android.Media.MediaPlayer to view and manage the audio/video playback.

Windows Phone 7

Microsoft.Xna.Framework.Audio.SoundEffect object manage sound from stream or file

1.2 Capturing Video (camera)

+ face recognition, capturing an image, insert into the communication process etc.

Android

The same as previous MediaRecorder object.

Windows Phone 7

Must be used FileSink Object and IsolatedStorageFileStream Object for storing into a file and VideoCaptureDevice and CaptureSource objects for specifying device and source of video.

Playing video

Android

MediaPlayer object is needed.

Windows Phone 7

MediaPlayerLauncher object controls source, visual controls, functions, orientation etc.

2. Contacts and calendar

This is the so called Personal Information Manager (PIM) software.

- **contact list;**
- **tasks;**
- **calendar.**

Accessing Contacts

Android

Android.Provider.ContactsContract object. **A generic table is used and each raw has a data that determines the values each of the columns hold.**

Before access contact record, you must insert permissions to the AndroidManifest.xml file:

```
<uses-permission android:name=“android.permission.READ_CONTACTS”/>
```

Windows Phone 7

Microsoft.Phone.UserData.Contacts class. **This performs asynchronous search witch results in an event handler:**

```
Contacts cons = new Contacts();
cons.SearchCompleted += new EventHandler<ContactsSearchEventArgs>
                                        (Contacts_SearchCompleted);
```

…

3. Messaging and Communication

Initiating a Voice call

Android

The user still needs to confirm the action before an application can initiate a phone call:

```
String uti = string.Format(“tel : {0}”, “0887-342-245”);
Android.Net.Uri phoneNumber = Android.Net.Uri.Parse(uti);
StartActivity( new Intent(Intent.ActionDial, PhoneNumber));
```

Windows Phone 7

You have to start a task : PhoneCallTask constructor with assigned prone number to a property. Confirmation before dialing is not mandatory (only when Show() exists):

```
PhoneCallTask task = new PhoneCallTask();
task.PhoneNumber = “887-333-234”;
task.Show();
```

4. Geo-Location

Getting GPS Location Information
Generally you have to set up a GPS listener in you application code and reference the listener's cashed data when the view needs the information.

Android

Access to all device sensors is achieved through SensorManager class.
You are able to create Sensor and connect listener to it, describing the functionality into a corresponding OnSensorChanged function.

Windows Phone SDK

Class Compass is used for the purpose. You can configure it (for example intervals). Event handler is used to read

```
_compass.CurrentValueChanged +=
          new EventHandler<SensorReadingEventArgs<CompassReading>>(…);
```

5. Accelerometer

Needed in games, to control navigation on the screen, to control scrolling speed in reading apps, to replace part or all the window.

All the info accelerometer API returns is X,Y,Z values

Android

Class **SensorManager** is used (the same as for GPS) with SensorType.Accelerometer .
A listener must be installed to read and evaluate the new values (RegisterListener(..);).
Handler function (OnSensorChanged(..) {} must be coded.

Windows Phone

Class **Accelerometer** is used. Event handler is wired to its instance .

Mobile programming concepts

1

Until 2010

Windows Mobile

(Visual Studio and Windows Mobile SDK)

-Usefull to start to develop for Windows® phones by using the same tools that developers use for desktop development:

Microsoft Visual Studio 2008;

Windows Mobile specific Microsoft .NET Compact Framework

-.NET Compact Framework (.NET CF) is a subset of the full Microsoft .NET Framework, with all the functionality that you must have to deliver powerful enterprise applications for the Windows Mobile platform.

Visual Studio from 2008 has built-in support for the .NET Compact Framework

Developers of desktop applications that use **Visual Studio** have all the necessary tools and knowledge to develop solutions for Windows Mobile platform.

There are many differences between the desktop and mobile platform.
However, the programming concepts remain the same.

You have to answer the following questions before you start to develop for a mobile:

- **What development tools do you have and want to use?**
- **What platform do you want to target?**

Other business-related considerations can also influence your decisions:

- **Do you want to create offline or online solution?**
- **How expensive is your connection?**
- **Do you want touch screen functionality?**
- **Are you targeting multiple or a single platform?**
- **What are your security requirements?**

History :

Visual Studio has supported Windows Mobile application development since Visual Studio 5.0.

Visual Studio 2005 supports development for Smartphone 2003, Pocket PC 2003 SE, and Windows CE. You can easily add support for Windows Mobile 5.0 (Smartphone, Pocket PC, and Pocket PC Phone Edition) and Windows Mobile 6 (Standard, Classic, and Professional) by installing SDKs for these platforms. SDKs include various tools that greatly improve your development experience, emulators, GPS, mobile operator emulator, and more.

Visual Studio 2008 brought some new features and functionalities, both in the IDE and in functionality.

The following list shows some new features in Visual Studio 2008:

- Support for unit testing, in Visual Studio Developer Edition and Visual Studio Team Suite
- Remote Performance Monitor
- Device Security Manager, part of Visual Studio 2008 IDE
- New Device Emulators and Device Emulator Manager
- New and redesigned New Project Wizard for creating new Windows Mobile projects

.NET CF (compact framework) is smaller than the .NET Framework, and supplies a subset of the .NET Framework functionality. Therefore, most applications that you develop for .NET CF should run without any modifications on the desktop .NET Framework platform.
There are two main reasons for reduced functionality in .NET CF:

- unsupported desktop features and
- limited storage space on mobile devices.

some new controls that are added and improved:

- MonthCalendar.	New in .NET CF 2.0.
- DataGrid.	Greatly improved.
- DateTimePicker.	New in .NET CF 2.0.
- LinkLabel.	New in .NET CF 2.0.
- Splitter.	New in .NET CF 2.0.
- WebBrowser.	New in .NET CF 2.0.

Because Visual Studio 2005 and 2008 support application development for various resolutions and various orientations, .NET CF 2.0 introduced also:
- Facilities to change Orientation from Portrait to Landscape or vice versa;
-Automatic scrollbars. If because of an orientation change, some controls remain outside the visible area, horizontal and vertical scrollbars are created automatically to provide access to all parts of user interface.
- Tab order support. Enables moving from one control to another by using the TAB key.

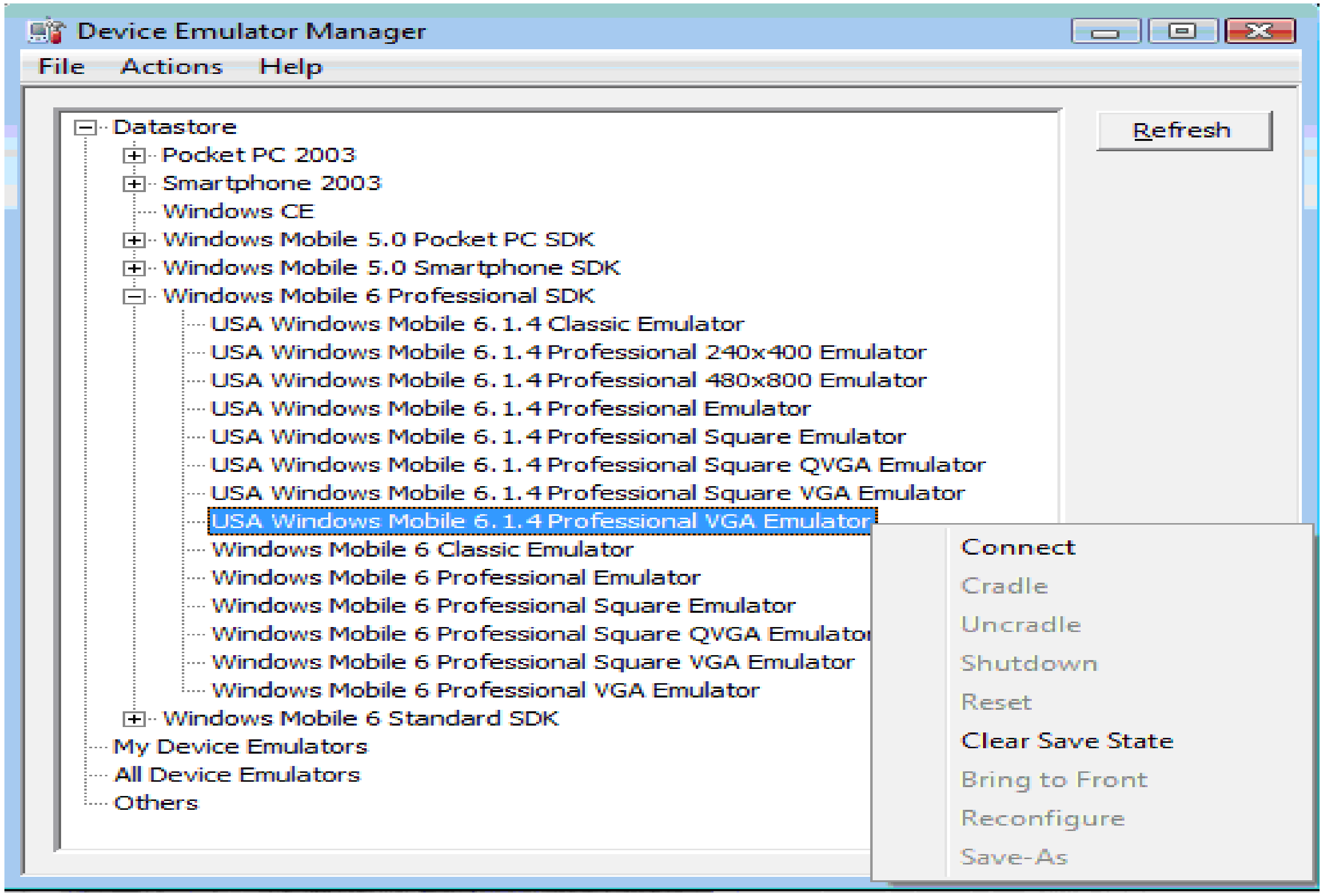
Working with data in .NET CF resembles working with data in the .NET Framework. ADO.NET is fully supported. Working with Microsoft SQL Server Compact Edition (various versions), and working with XML, Web services, and files as data stores is supported. .NET CF supports datasets, data binding, and collections.

Windows Mobile SDK Tools

Windows Mobile SDK Tools are an addition to Visual Studio and .NET CF. They provide support and templates for developing applications targeting Windows Mobile platforms.
Windows Mobile SDK Tools are free downloads. They require you to have Visual Studio 2005 or 2008 and .NET CF installed. There are two separate installations for two platforms. One is Windows Mobile Standard (previously called Smartphone) and the other one is Windows Mobile Professional, targeted for both Classic (previously called Pocket PC) and Professional (previously called Pocket PC Phone Edition).

Device Emulator Manager

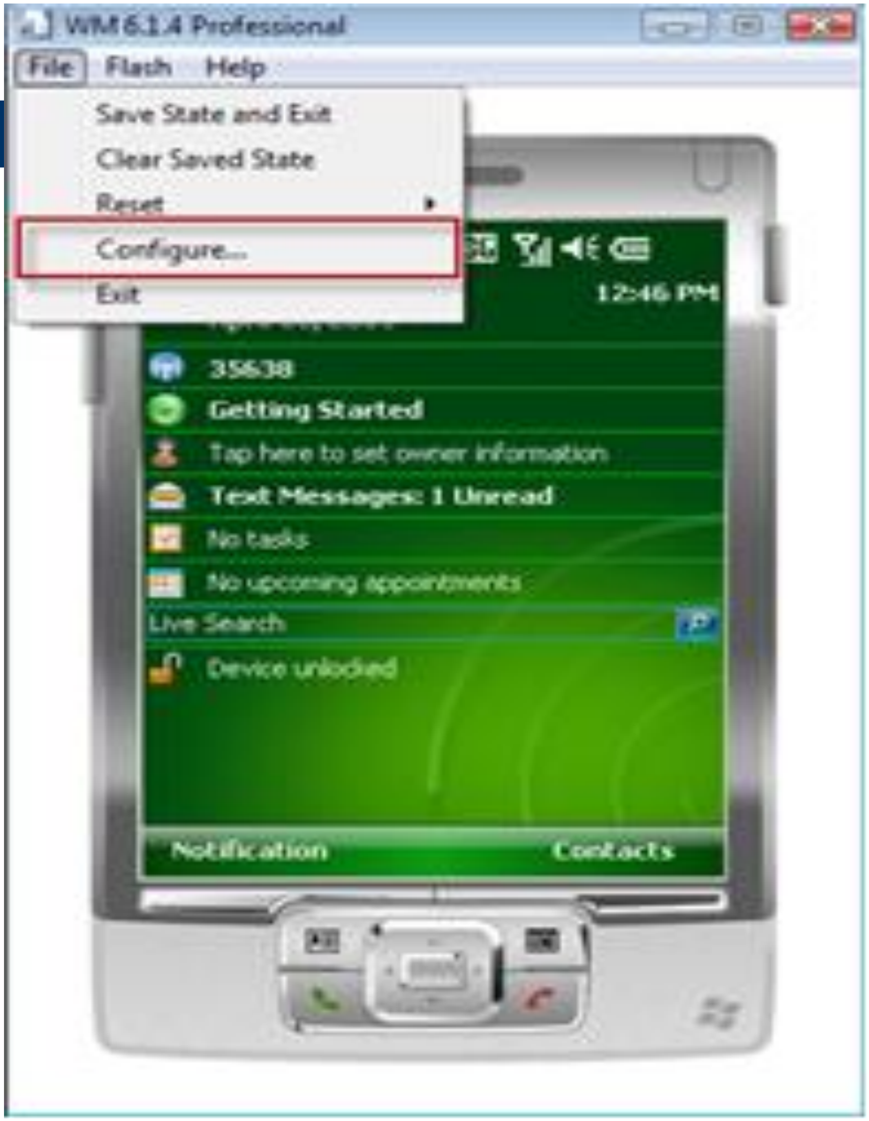
Device Emulator Manager manages Device Emulators that are installed on the developer workstation



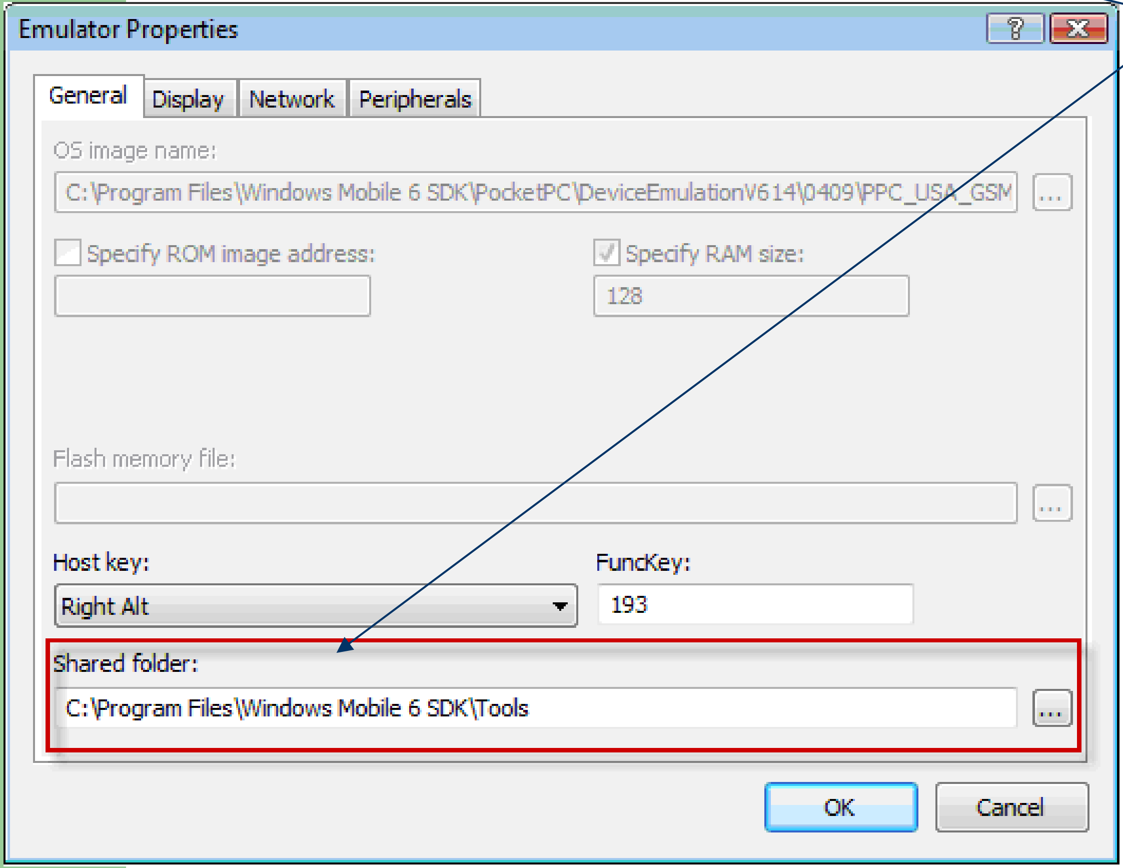
```
Datastore
  Pocket PC 2003
  Smartphone 2003
  Windows CE
  Windows Mobile 5.0 Pocket PC SDK
  Windows Mobile 5.0 Smartphone SDK
  Windows Mobile 6 Professional SDK
    USA Windows Mobile 6.1.4 Classic Emulator
    USA Windows Mobile 6.1.4 Professional 240x400 Emulator
    USA Windows Mobile 6.1.4 Professional 480x800 Emulator
    USA Windows Mobile 6.1.4 Professional Emulator
    USA Windows Mobile 6.1.4 Professional Square Emulator
    USA Windows Mobile 6.1.4 Professional Square QVGA Emulator
    USA Windows Mobile 6.1.4 Professional Square VGA Emulator
    USA Windows Mobile 6.1.4 Professional VGA Emulator
    Windows Mobile 6 Classic Emulator
    Windows Mobile 6 Professional Emulator
    Windows Mobile 6 Professional Square Emulator
    Windows Mobile 6 Professional Square QVGA Emulator
    Windows Mobile 6 Professional Square VGA Emulator
    Windows Mobile 6 Professional VGA Emulator
  Windows Mobile 6 Standard SDK
My Device Emulators
All Device Emulators
Others
```

Device Emulators

You can start the Device Emulator from Microsoft Visual Studio IDE, manually or automatically when you deploy the application, or from Device Emulator Manager. When it is up and running, you can configure it to meet your needs. You can access configuration options on the Device Emulator menu.



You frequently have to access host files and folders from the Device Emulator. The following illustration shows how to set up the location of a Shared folder in the Emulator configuration.
.
That gives you access to that folder from the Device Emulator. The Shared folder is displayed as a Storage Card in the Emulator File Explorer.



Emulator Properties

General | Display | Network | Peripherals

OS image name:
C:\Program Files\Windows Mobile 6 SDK\PocketPC\DeviceEmulationV614\0409\PPC_USA_GSM

Specify ROM image address:

[✓] Specify RAM size:
128

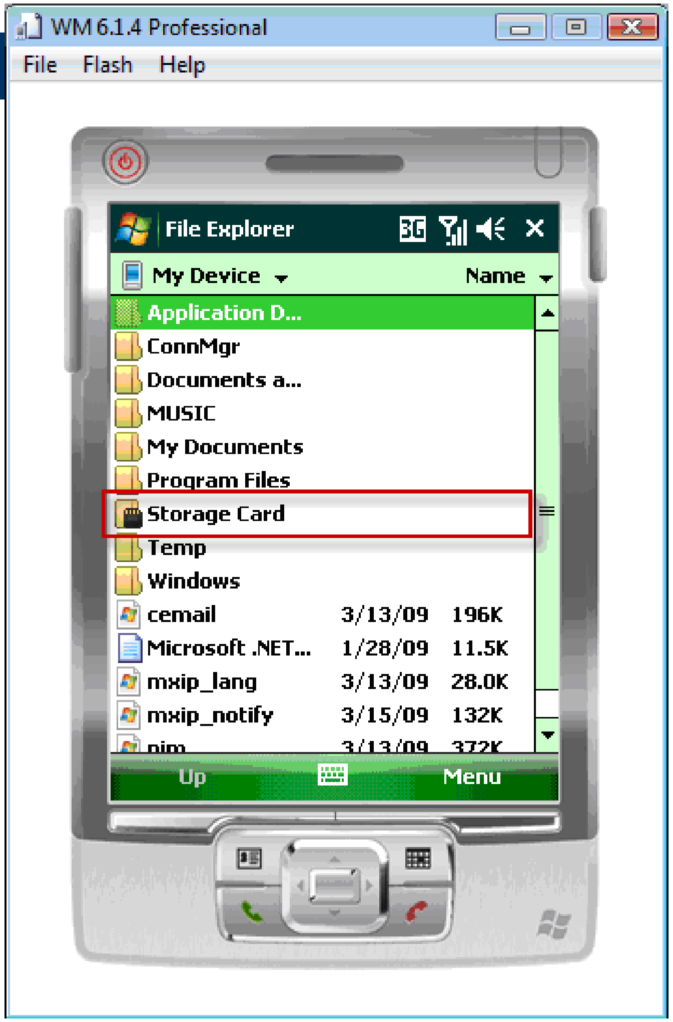
Flash memory file:

Host key:
Right Alt

FuncKey:
193

Shared folder:
C:\Program Files\Windows Mobile 6 SDK\Tools

OK | Cancel



WM 6.1.4 Professional

File | Flash | Help

File Explorer

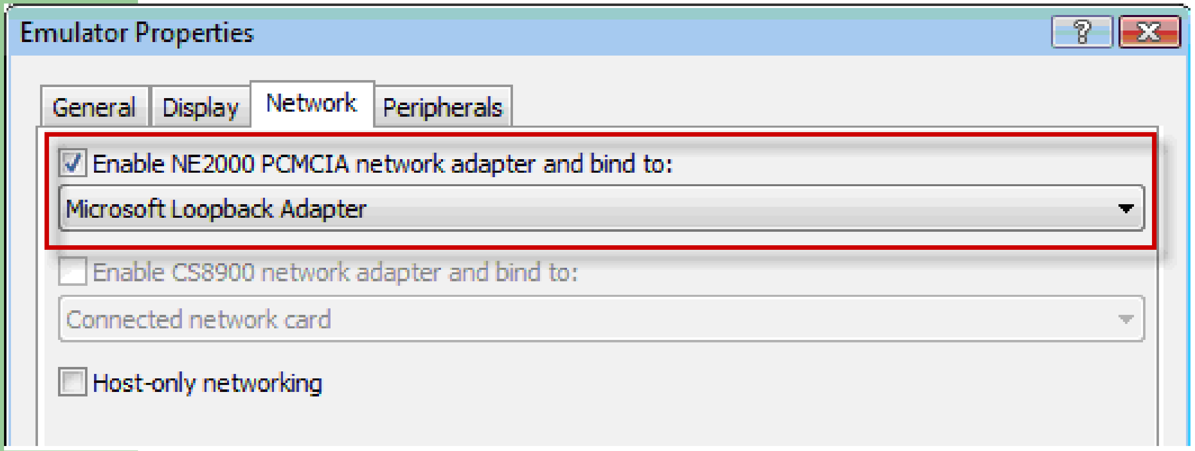
My Device | Name

Application D...		
ConnMgr		
Documents a...		
MUSIC		
My Documents		
Program Files		
Storage Card		
Temp		
Windows		
cemail	3/13/09	196K
Microsoft .NET...	1/28/09	11.5K
mxip_lang	3/13/09	28.0K
mxip_notify	3/15/09	132K
pim	3/13/09	372K

Up | Menu

network connection in the Device Emulator

To simulate real-world environment, sometimes you must have a network connection in the Device Emulator. To do that, you have to set up the host network adapter that you want to connect to in the emulator.



Emulator Properties

General | Display | Network | Peripherals

[✓] Enable NE2000 PCMCIA network adapter and bind to:

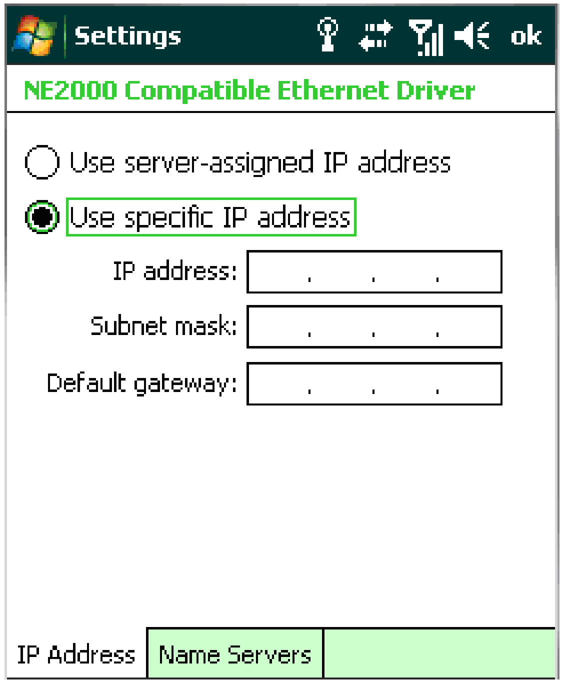
Microsoft Loopback Adapter

[ ] Enable CS8900 network adapter and bind to:

Connected network card

[ ] Host-only networking

After the network adapter is set up, you have to set up the network adapter in the emulator :



Settings ok

NE2000 Compatible Ethernet Driver

○ Use server-assigned IP address

● Use specific IP address

IP address: . . .

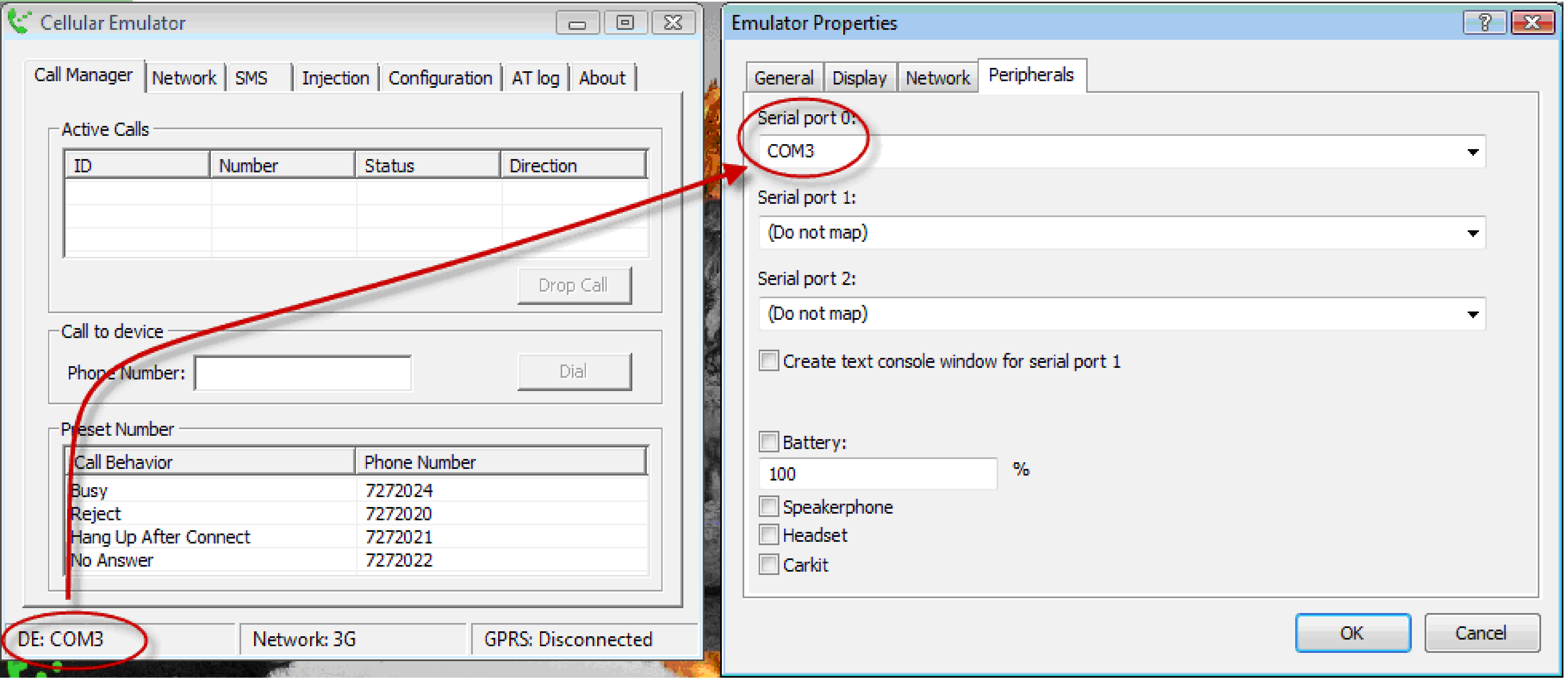
Subnet mask: . . .

Default gateway: . . .

IP Address | Name Servers

Cellular Emulator

Cellular Emulator is part of Windows Mobile SDK Tool. It emulates the presence of mobile operator network that enables developers to test phone functionality in their applications from a Device Emulator. To configure Cellular Emulator to communicate with Device Emulator, you have to map its COM port to the physical port of Device Emulator.



Active Calls

ID	Number	Status	Direction

Call to device

Phone Number:

Preset Number

Call Behavior	Phone Number
Busy	7272024
Reject	7272020
Hang Up After Connect	7272021
No Answer	7272022

DE: COM3 | Network: 3G | GPRS: Disconnected

Serial port 0: COM3

Serial port 1: (Do not map)

Serial port 2: (Do not map)

☐ Create text console window for serial port 1

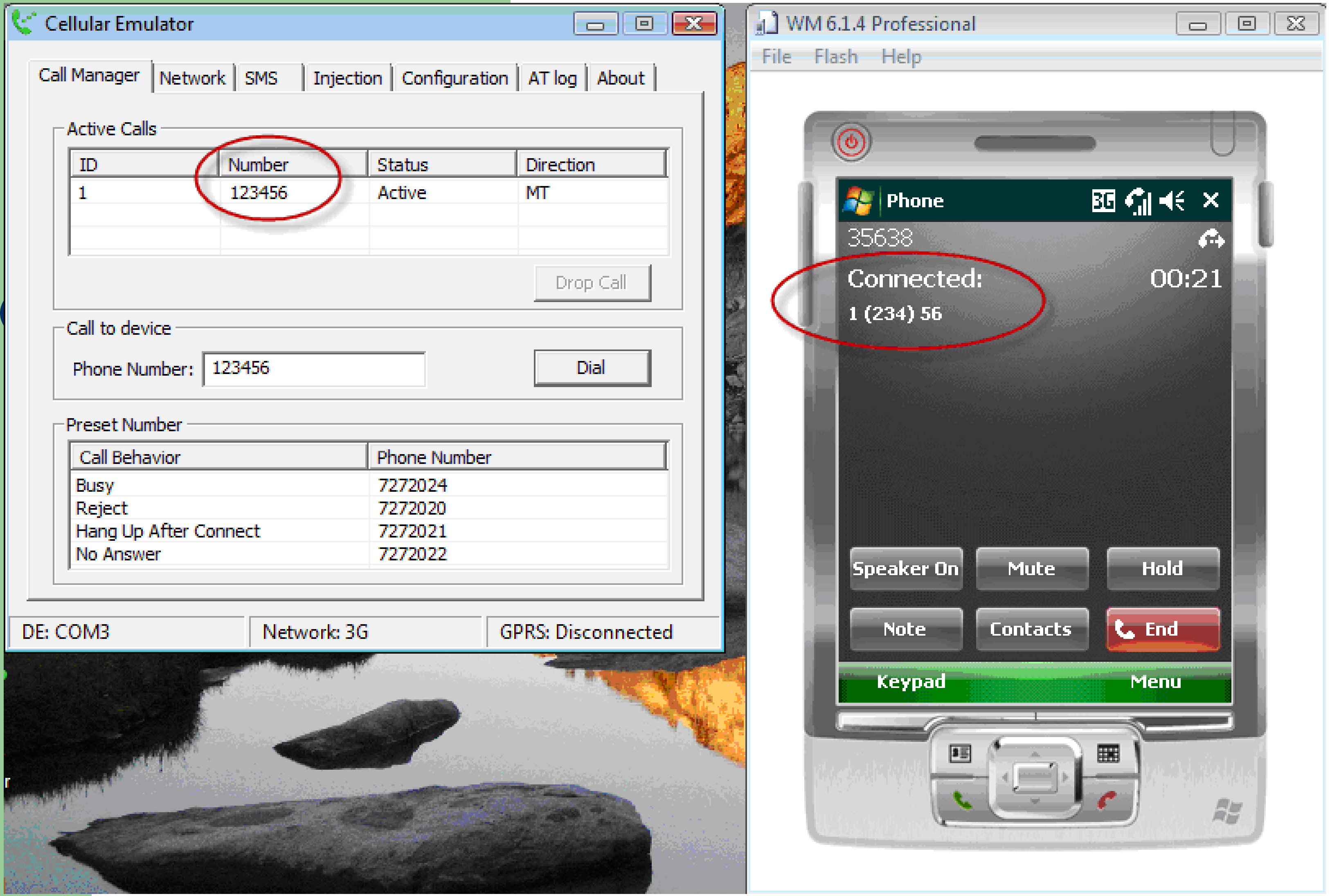
☐ Battery: 100 %

☐ Speakerphone

☐ Headset

☐ Carkit

When it is run and configured, the Cellular Emulator enables Windows Mobile Professional Device Emulator to send and receive phone calls and SMS messages, and data services, such as GPRS and 3G.



Active Calls

ID	Number	Status	Direction
1	123456	Active	MT

Call to device

Phone Number: 123456

Preset Number

Call Behavior	Phone Number
Busy	7272024
Reject	7272020
Hang Up After Connect	7272021
No Answer	7272022

DE: COM3 | Network: 3G | GPRS: Disconnected

35638

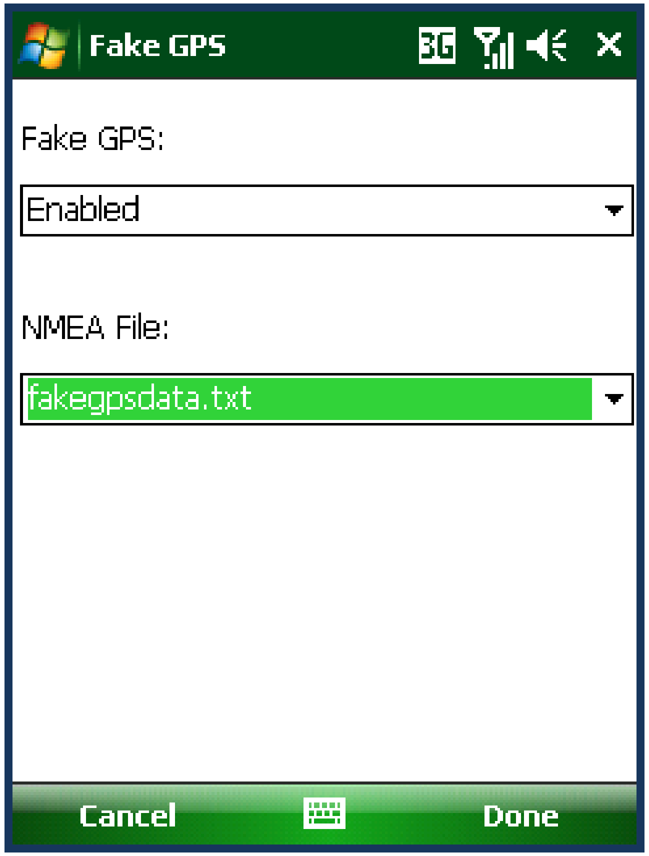
Connected: 00:21

1 (234) 56

Fake GPS

Fake GPS enables you to use GPS functionalities from Device Emulators or physical devices that do not have GPS, or to test applications in scenarios where you do not have GPS coverage.

Fake GPS is a small application that you have to install with Device Emulator. When it is enabled, it runs like a service on the emulator. It randomly provides GPS coordinates from one of the files that are included to applications that use the assembly. The application behaves the same way it will with a physical GPS on a device and reads real GPS coordinates from a GPS device.



Fake GPS:

Enabled

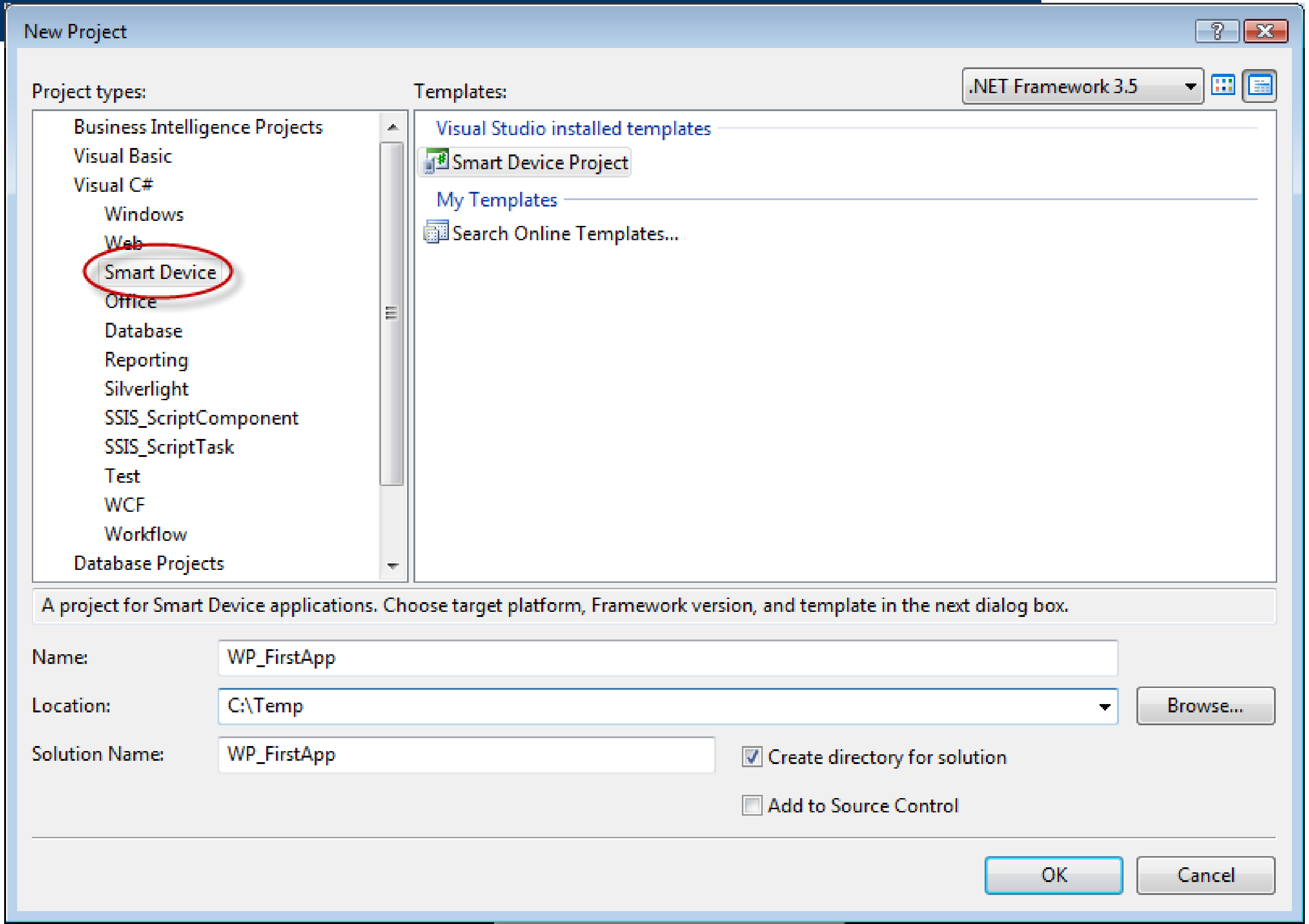
NMEA File:

fakegpsdata.txt

You have to install the following software products and tools to your development environment in order to get started:

- Visual Studio 2008 or later
    - Latest Visual Studio, .NET Framework and .NET Compact Framework updates
    - Windows Mobile SDK Tools Standard or Professional, or both
    - Latest emulators to be supported, not included in SDK Tools
    - SQL Server to work with local data stores

- You are ready to start and develop your first Windows Mobile solution.


New Project

Project types:

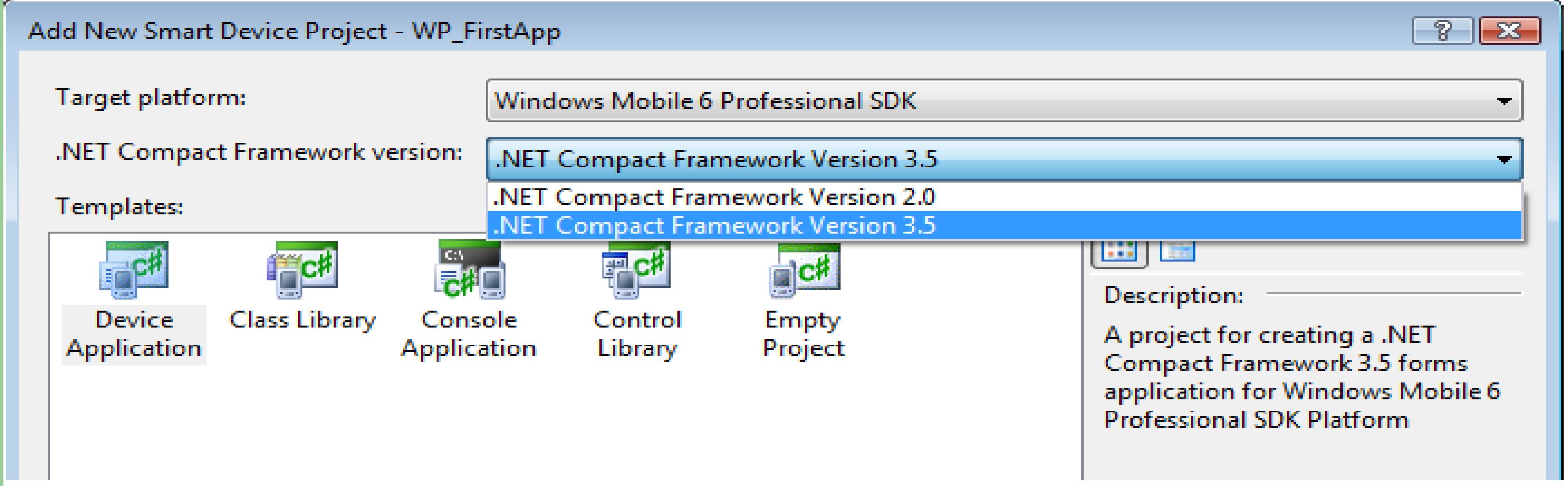
Business Intelligence Projects
Visual Basic
Visual C#
Windows
Web
Smart Device
Office
Database
Reporting
Silverlight
SSIS_ScriptComponent
SSIS_ScriptTask
Test
WCF
Workflow
Database Projects

Templates: .NET Framework 3.5

Visual Studio installed templates
Smart Device Project
My Templates
Search Online Templates...

A project for Smart Device applications. Choose target platform, Framework version, and template in the next dialog box.

Name: WP_FirstApp
Location: C:\Temp
Solution Name: WP_FirstApp
☑ Create directory for solution
☐ Add to Source Control


Add New Smart Device Project - WP_FirstApp

Target platform: Windows Mobile 6 Professional SDK

.NET Compact Framework version: .NET Compact Framework Version 3.5

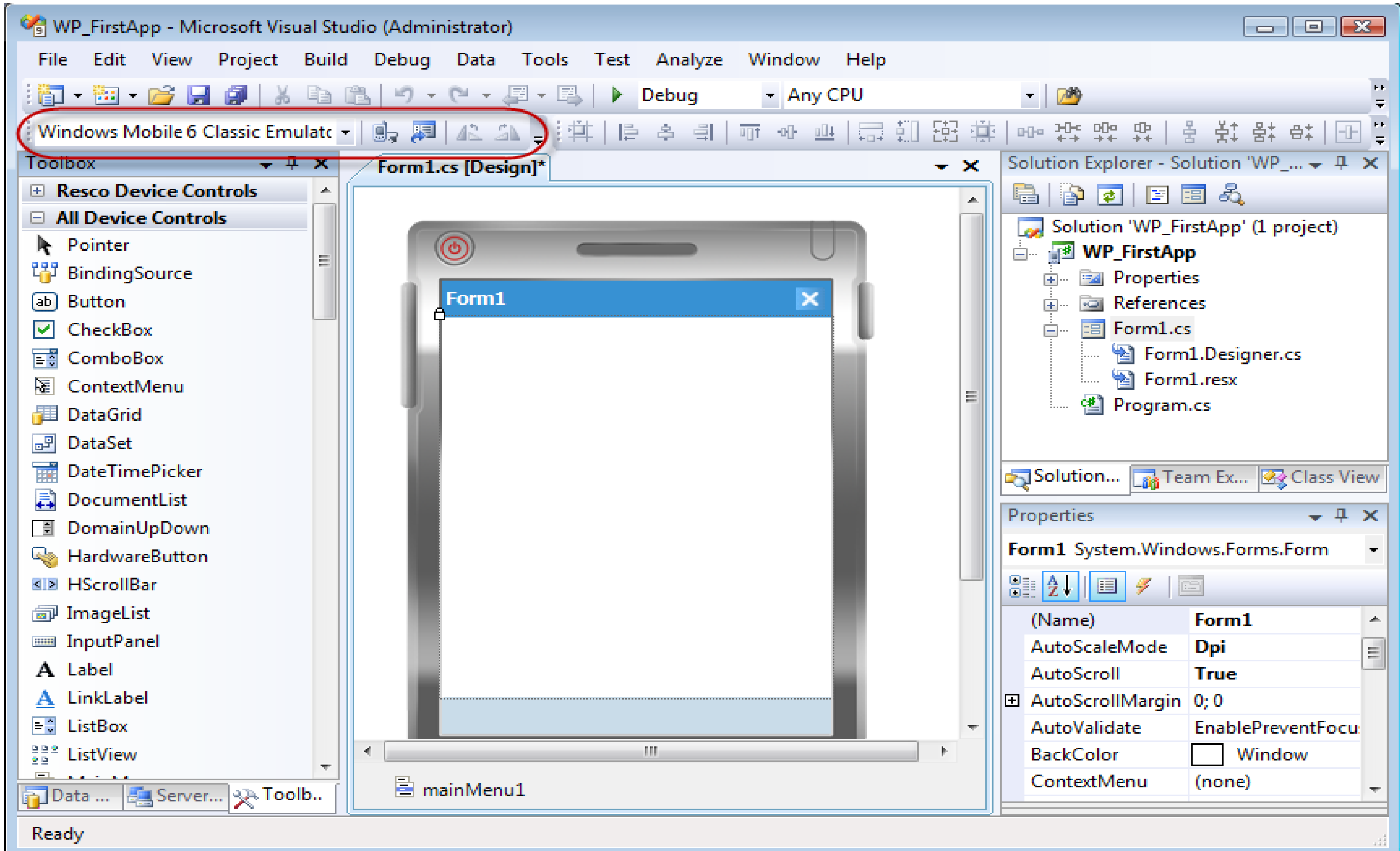
.NET Compact Framework Version 2.0
.NET Compact Framework Version 3.5

Templates:

Description:

A project for creating a .NET Compact Framework 3.5 forms application for Windows Mobile 6 Professional SDK Platform

Now you have a regular Visual Studio solution with a basic application skeleton


WP_FirstApp - Microsoft Visual Studio (Administrator)

(Name)	Form1
AutoScaleMode	Dpi
AutoScroll	True
AutoScrollMargin	0; 0
AutoValidate	EnablePreventFocu
BackColor	Window
ContextMenu	(none)

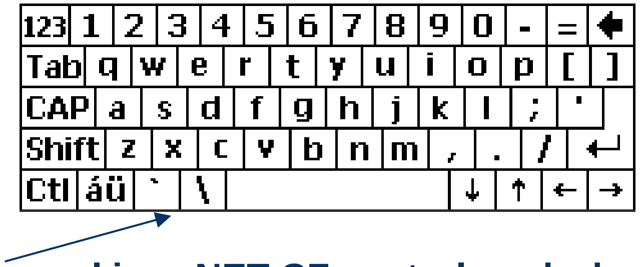
Handling inputs

Compared to standard desktop input methods, keyboard and mouse, mobile devices handle input somewhat differently. **Keyboard** became a standard option in many mobile devices so that users can use it as an input device for mobile devices, but more common options are **Taps, Software Input Panel (SIP), and Hardware keys.**

Tap is to a mobile device, what the left mouse double-click is to the desktop. It is a common input method on touch-screen devices. **Tap-and-hold** corresponds to right-click. It opens the shortcut menu for selected items. When tap is used on a control in a Windows form, it starts all corresponding events for that control: MouseDown, MouseUp, MouseMove, Click, DoubleClick.

From a programmatic perspective, it is the equivalent to a mouse click, and you can handle all these events in your application just as you handle them in desktop applications.

The **soft input panel** lets users accept keyboard input without a physical keyboard. SIP in combination with Tap provides a real keyboarding experience for users, invoking events associated with pressing keys on the keyboard: KeyUp, KeyDown, and KeyPress.



The Input panel is a .NET CF control, and when you add it to the form, it can be programmatically controlled from the code. You can see it when the text box control receives the focus:

```
inputPanel1.Enabled = true;
```

Error handling

Error handling in .NET CF is performed exactly like in the .NET Framework, by using a
try….
    catch….
finally …
                structure

Working with data

.NET CF supports working with various data sources. The concept is the same as the .NET Framework and most important, ADO.NET is supported in most of its functionality. Windows Mobile applications can use SQL Server CE, XML, SQL Server, Web services as data sources.

XML is a common format that most data sources can export data to, and import data from. Even though it can be used by most data sources, we do not recommend it as a data store, because it has a large overhead, it is not optimized, and is much slower than a relational database, because it has no indexes. However, if there is no other option, XML can be used as a data store on the device.
SQL Server can directly be accessed from devices as a data store.

Web Services service are a good choice for transferring data in the n-tier environment when you do not have direct access to SQL Server (either for direct data access or for replication).

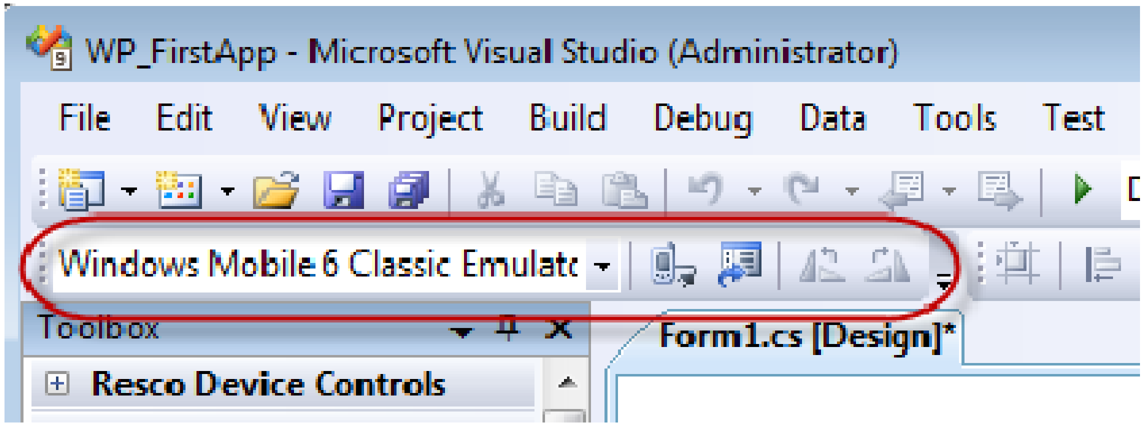
Deployment and packaging

The last step in the project development cycle is to deploy the completed solution to the device and to package it for distribution and mass deployment.
There are several ways with which an application can be deployed to the device or emulator:

Development IDE installer;
Device Installer;

Development IDE installer

The easiest way to deploy the application is when the device (beeing open for deployment) is connected to the development workstation through ActiveSync or WMDC.
The first step is to select the deployment device or emulator.



When you select it, you can connect to the target device manually from the toolbar or the deployment process connects automatically if it is not connected already.

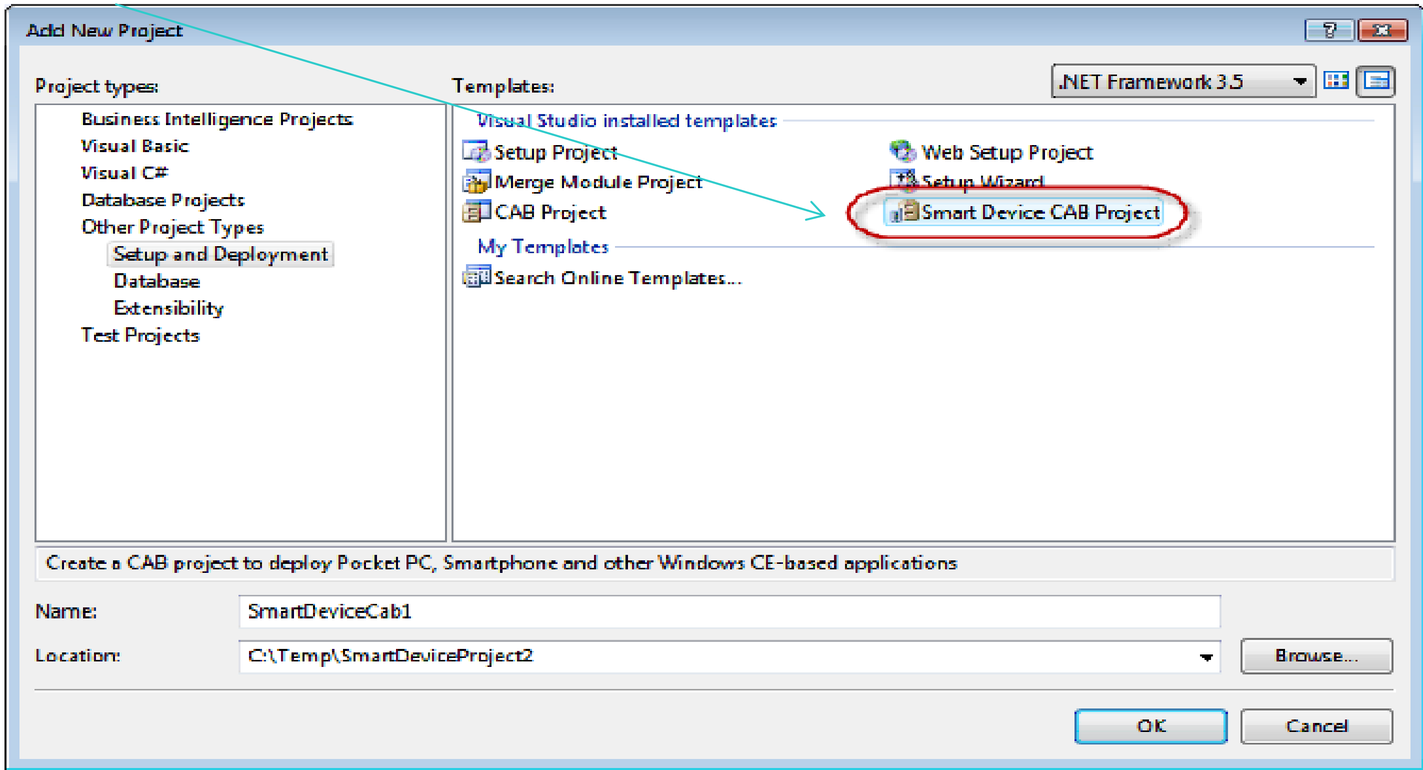
Device installer

It is not very likely that every device that is used in production arrives at the developer's desk for application deployment.

Device Installer is the method that enables applications to be easily deployed directly on the device, without any need for a connection between the device and the developer workstation. All assemblies and dependencies are packed in a single (or more) **CAB file** which is copied to the device and executed there. The CAB execution on the device installs it with predefined parameters. The CAB file can be distributed through e-mail, a Web server, external cards, or by coping it through

ActiveSync

In the **Add New Project dialog box**, under Templates, there is a project template in Visual Studio to enables you to create CAB projects:

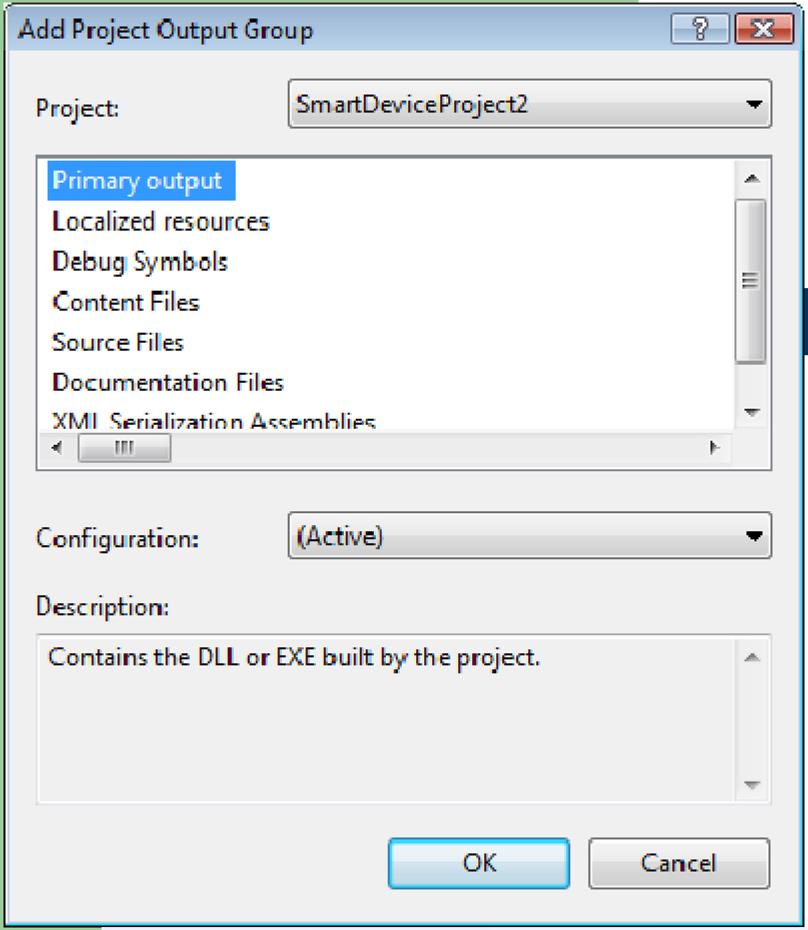


Create a CAB project to deploy Pocket PC, Smartphone and other Windows CE-based applications

Name: SmartDeviceCab1

Location: C:\Temp\SmartDeviceProject2

In the CAB project, you have to define project output, which is the application that you just finished


Add Project Output Group

Project: SmartDeviceProject2

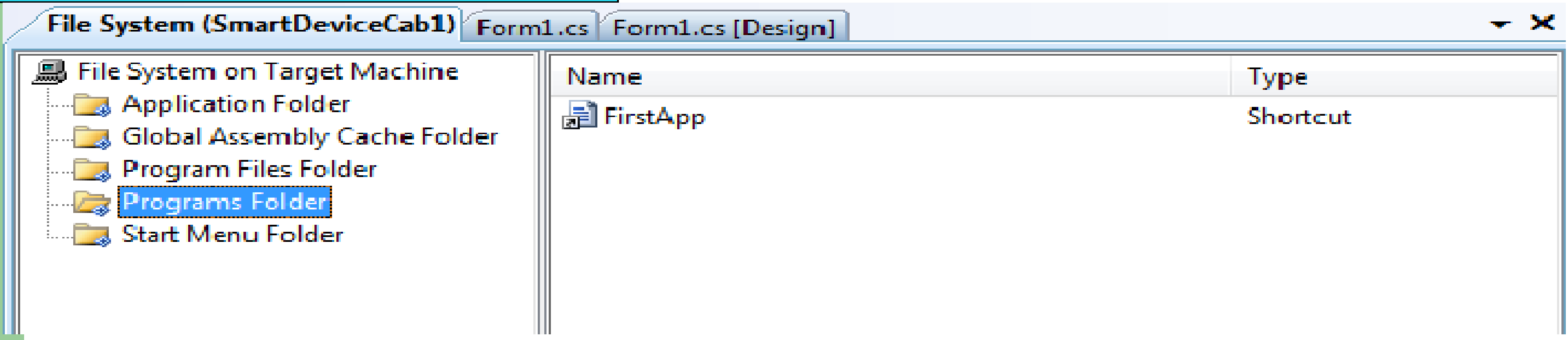
Primary output
Localized resources
Debug Symbols
Content Files
Source Files
Documentation Files
XML Serialization Assemblies

Configuration: (Active)

Description:

Contains the DLL or EXE built by the project.

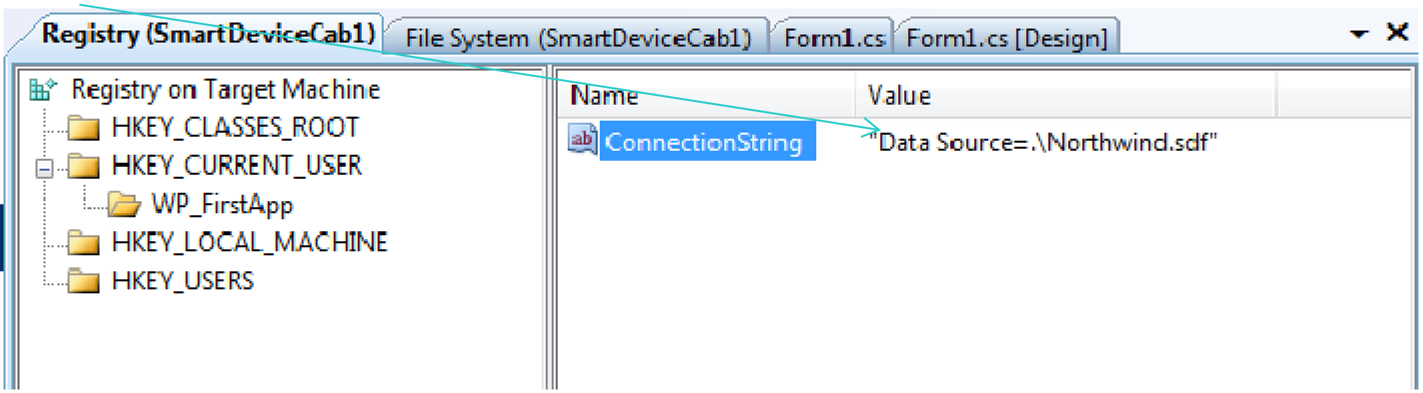
After the project output is set, it is automatically added to the application folder for installation


File System (SmartDeviceCab1) | Form1.cs | Form1.cs [Design]

File System on Target Machine
- Application Folder
- Global Assembly Cache Folder
- Program Files Folder
- Programs Folder
- Start Menu Folder

Name	Type
FirstApp	Shortcut

Through the Deployment project, you can also alter registry settings and add new keys for your application

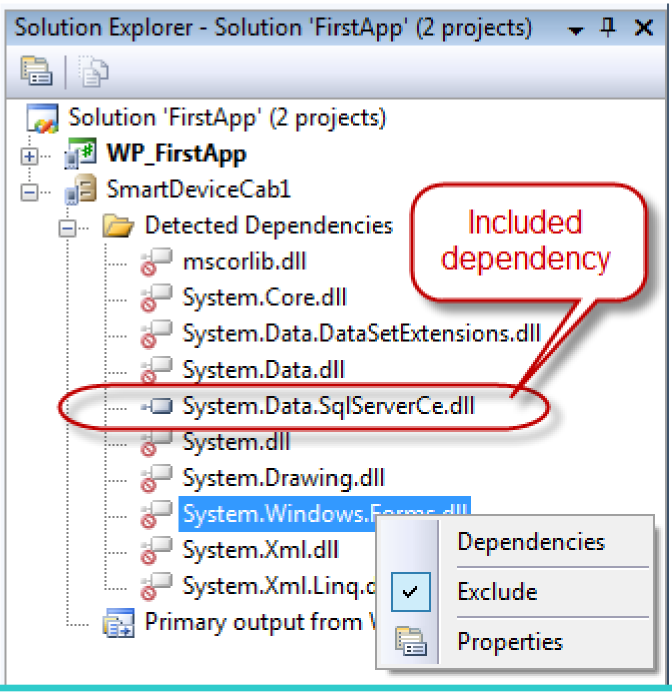


Registry (SmartDeviceCab1) | File System (SmartDeviceCab1) | Form1.cs | Form1.cs [Design]

- Registry on Target Machine
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
    - WP_FirstApp
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS

Name	Value
ConnectionString	"Data Source=.\Northwind.sdf"

Even though it is good practice to distribute and install dependencies (.NET CF, SQL Server CE) separately to keep the CAB file as small as possible, you can also include dependencies into the final CAB. By default, they are excluded:



Solution Explorer - Solution 'FirstApp' (2 projects)

- Solution 'FirstApp' (2 projects)
  - WP_FirstApp
  - SmartDeviceCab1
    - Detected Dependencies
      - mscorlib.dll
      - System.Core.dll
      - System.Data.DataSetExtensions.dll
      - System.Data.dll
      - System.Data.SqlServerCe.dll
      - System.dll
      - System.Drawing.dll
      - System.Windows.Forms.dll
      - System.Xml.dll
      - System.Xml.Linq.d
    - Primary output from W

When all parameters for the Deployment project are set, you can build it. The result of the building action is a redistributable, self-executing CAB file, ready to be distributed and deployed directly on devices.

Pocket outlook object model (POOM)

The POOM API is part of the Microsoft.WindowsMobile.PocketOutlook library. Through POOM, you can access Calendar, Contacts and Tasks on the device.

All activities that use POOM functionality are based on the OutlookSession class. Therefore, in your applications, you must always have an instance of that object at the application level.
The following code example shows that an OutlookSession object is declared the first time that it appears and that an instance of it is created.

```
using Microsoft.WindowsMobile.PocketOutlook;
namespace WP_FirstApp
{
    public partial class Form1 : Form
    {
        private OutlookSession outlook;

        private void Main_Load(object sender, EventArgs e)
        {
            outlook = new OutlookSession();
```

The following code example shows that it is very easy to create a new task from the application.

```
Task task = new Task();
 task.Subject = textBox1.text;
 task.Body = textBox2.text;
 outlook.Tasks.Items.Add(task);
```

New task will be added to Pocket Outlook Tasks, and can be managed either from code or from Tasks menu-option directly. Existing tasks can also be altered from the application.

You can also manage Contacts from the application, using the same parent object as for Tasks. When you create a new contact, all fields are optional. However, some basic set of information should be set, to recognize the contact later.
The following code example shows the code for creating a new contact.

```
 Contact con = outlook.Contacts.Items.AddNew();
Con.FirstName = textBox1.text;
Con.LastName = textBox2.text;
Con.Update();
```

To select an existing contact, use the ChooseContactDialog object. This object can be used to access a specific contact or a specific property of a contact. ChooseContactDialog enables you to filter contacts to be shown to the user, by using the RestrictContacts property and setting the filter. The following code example shows how to use an instance of the ChooseContactDialog class to present a Contacts dialog box to the user and how to retrieve the business telephone number of the contact if a contact was selected by the user from the dialog box.

```
ChooseContactDialog dialog = new ChooseContactDialog();
dialog.RestrictContacts = "[Department] = \"Field\"";
dialog.Title = "Assign To";
if (dialog.ShowDialog() == DialogResult.OK)
            {
            str = dialog.SelectedContact.BusinessTelephoneNumber;
             }
```

E-mail

E-mail functionality is accessed through the **EmailMessage object.** The EmailMessage object contains all standard properties that e-mail messages usually have: Attachments, Bcc, BodyText, CC, Importance, Subject, To and more.
The following code example show how to create a new e-mail message, set properties, and send an e-mail message to the contact previously selected.

```
EmailMessage email = new EmailMessage();
email.To.Add(new Recipient(dialog.SelectedContact.Email1Address));
email.Subject = textBox1.text;
email.BodyText = textBox2.text;
email.Send("ActiveSync");
```

Handling SMS

Handling SMS resembles handling e-mail. Except for having fewer properties, the main difference is that when it calls a **Send method**, an SMS message is sent immediately, whereas **an e-mail message waits for the next synchronization.**
The following code example shows how to create a new SMS message and send it.

```
SmsMessage sms = new SmsMessage();
sms.To.Add(new Recipient("123456"));
sms.Body = "WP_FirstApp SMS message";
sms.Send();
```

SMS interception

SMS messages can easily be intercepted by the application by using the MessageInterceptor object. The MessageInterceptor object is contained in the Microsoft.WindowsMobile.PocketOutlook.MessageInterception namespace. Messages to be intercepted can be filtered by the message body, the message subject or sender.
SMS Interception can be used even if the application is currently not running.
The following example code shows how to implement SMS interception.

```
// SMS Interception Setup
      smsInterceptor = new MessageInterceptor( InterceptionAction.NotifyAndDelete );
      smsInterceptor.MessageCondition = new MessageCondition(
                                MessageProperty.Sender,
                                MessagePropertyComparisonType.Equal, "+14254448851");
      smsInterceptor.MessageReceived += new MessageInterceptorEventHandler
                                                  (smsInterceptor_MessageReceived);

// SMS Interception event handler
private void smsInterceptor_MessageReceived(object sender,
                                                         MessageInterceptorEventArgs e )
          {
           if (e.Message.GetType() == typeof(SmsMessage))
                    {
                    SmsMessage message = (SmsMessage)e.Message;
                     //do something with message
….
```

Telephony

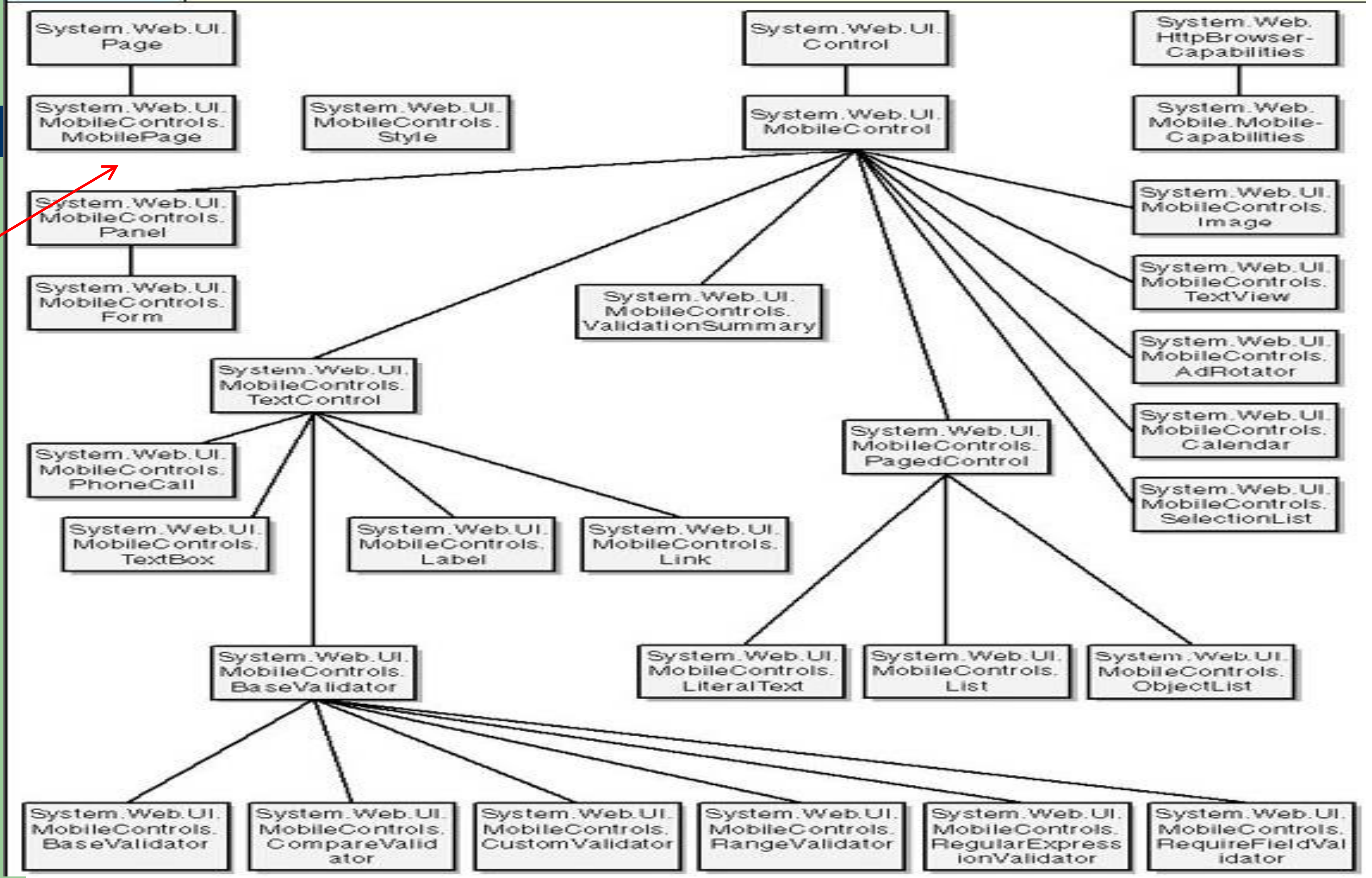
Making voice calls from applications enables you to use phone functionality directly from the application, without having to go to a device dialer. The Phone class is in the Microsoft.WindowsMobile.Telephony library. Use the Phone class to establish voice calls. The following example code shows how to establish voice calls by using the contact that you previously selected.

```
using Microsoft.WindowsMobile.Telephony;

private void MakeVoiceCall()
  {
            Phone phone = new Phone();
            phone.Talk(dialog.SelectedContact.BusinessTelephoneNumber);
   }
```

Mobile Internet Toolkit (.NET) Standard Controls

some of the standard controls of the Mobile Internet Toolkit:

Figure 5.1 *The top-level heirachy of the mobile controls*


System.Web.UI.Page
System.Web.UI.MobileControls.MobilePage
System.Web.UI.MobileControls.Style
System.Web.UI.Control
System.Web.UI.MobileControl
System.Web.HttpBrowser-Capabilities
System.Web.Mobile.Mobile-Capabilities
System.Web.UI.MobileControls.Panel
System.Web.UI.MobileControls.Form
System.Web.UI.MobileControls.ValidationSummary
System.Web.UI.MobileControls.Image
System.Web.UI.MobileControls.TextView
System.Web.UI.MobileControls.AdRotator
System.Web.UI.MobileControls.Calendar
System.Web.UI.MobileControls.SelectionList
System.Web.UI.MobileControls.TextControl
System.Web.UI.MobileControls.PhoneCall
System.Web.UI.MobileControls.TextBox
System.Web.UI.MobileControls.Label
System.Web.UI.MobileControls.Link
System.Web.UI.MobileControls.PagedControl
System.Web.UI.MobileControls.BaseValidator
System.Web.UI.MobileControls.LiteralText
System.Web.UI.MobileControls.List
System.Web.UI.MobileControls.ObjectList
System.Web.UI.MobileControls.BaseValidator
System.Web.UI.MobileControls.CompareValidator
System.Web.UI.MobileControls.CustomValidator
System.Web.UI.MobileControls.RangeValidator
System.Web.UI.MobileControls.RegularExpressionValidator
System.Web.UI.MobileControls.RequireFieldValidator

Structure of an application

A developer working in an object-oriented language such as C# or C++.
The .aspx file is the place for defining a class that is specific to ASP.NET.

In mobile Web applications, this class must descend—directly or indirectly—from the **System.Web.UI.MobileControls.MobilePage** class.
A mobile Web Forms page declares itself as a descendant of the MobilePage class through the @Page directives at the top of the .aspx file:

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="c#"%>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
                                                    Assembly="System.Web.Mobile" %>
```

If you have a code-behind module, this page must inherit from a class in the code-behind module, which must itself inherit from the MobilePage class.

In such instances, the @Page directive must specify the name of the code-behind module and the class within it:

```
<%@ Page Codebehind="Default.aspx.cs" Language="c#"
                                                    Inherits="MyMobileWebForm%>
```

the file Default.aspx :

```
<%@ Page Codebehind="Default.aspx.cs" Language="c#"    Inherits="MyMobileWebForm"
                                                AutoEventWireup="True"%>
<%@ Register TagPrefix="mobile"    Namespace="System.Web.UI.MobileControls"
                              Assembly="System.Web.Mobile" %>
<mobile:Form runat="server">
         <mobile:Label runat="server"/>
</mobile:Form>
```

means that methods such as Page_Load or Page_Init are "wired-up" automatically. Visual Studio .NET sets AutoEventWireup to False by default in the mobile Web Forms pages it creates and automatically inserts the code needed to wire up the Page events.

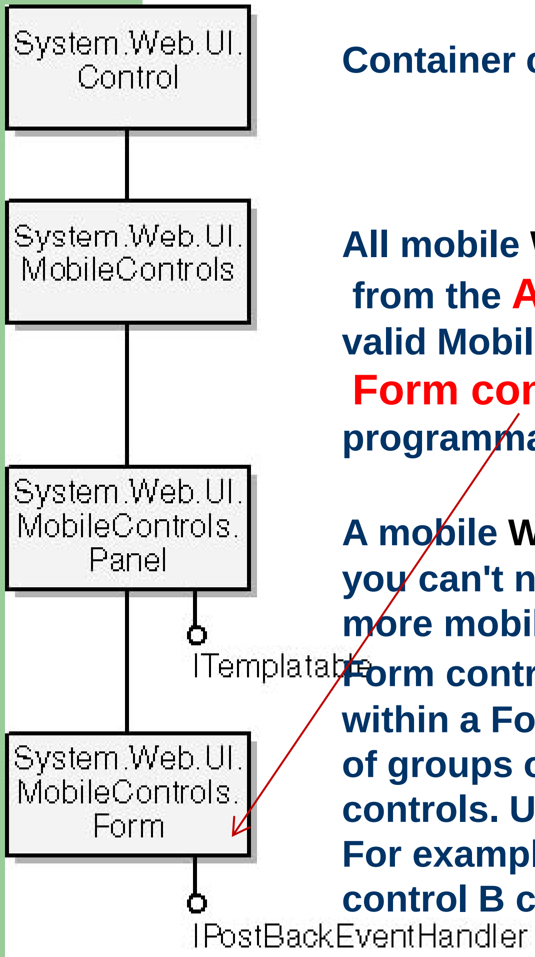
And, the Code-behind module Default.aspx.cs:

```
using System;
public class MyMobileWebForm : System.Web.UI.MobileControls.MobilePage
{
protected System.Web.UI.MobileControls.Label Label1;
private void Page_Load(object sender, System.EventArgs e)
         {      Label1.Text = "Hello World";    }
}
```

The XML text in Default.aspx represents the server control syntax for a Label control contained within a Form control.

When this code compiles, the runtime actually creates an instance of the MobilePage-derived class, which contains an instance of the Form class and in that, an instance of the Label class.

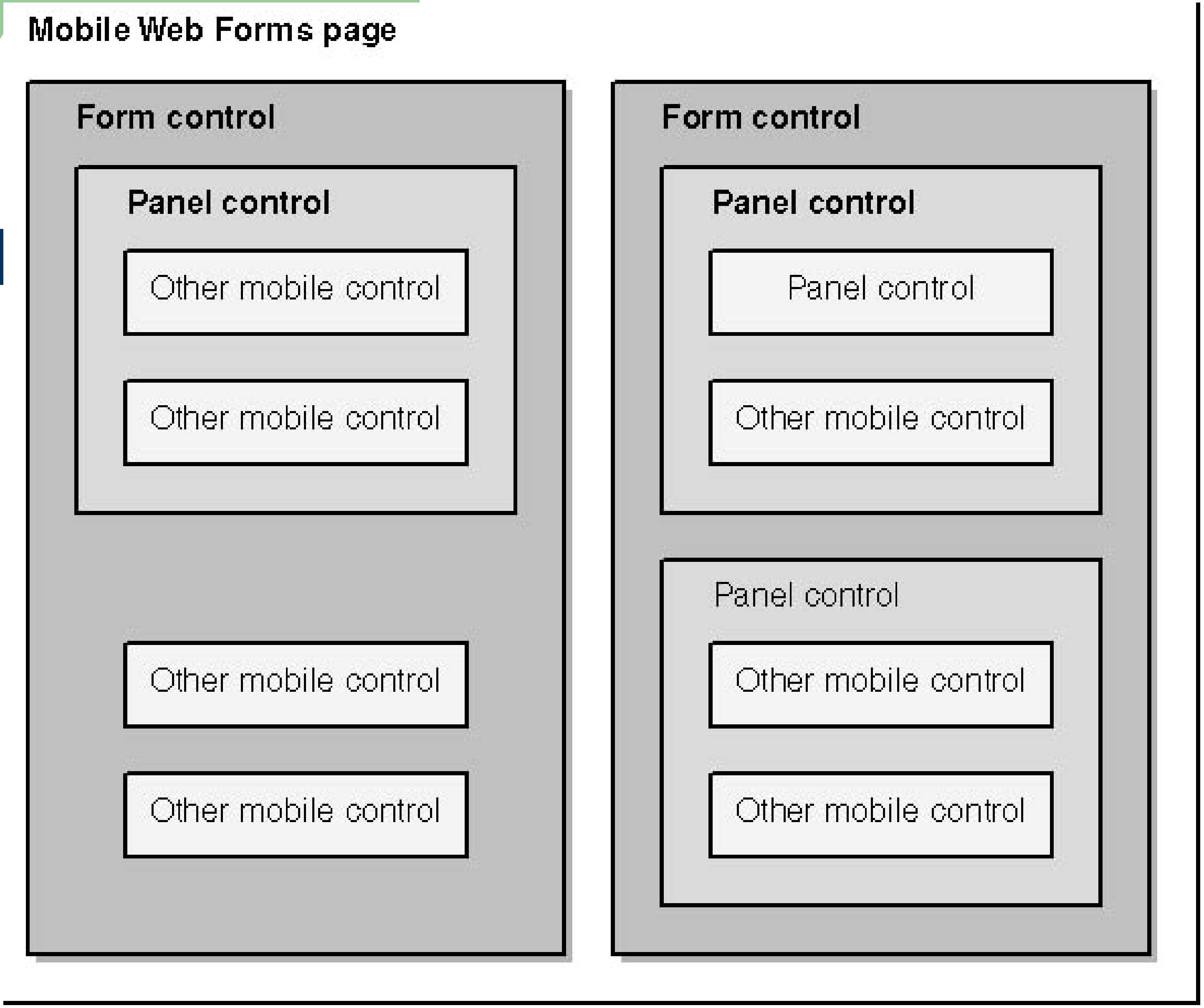
Container Controls



Container controls provide a powerful means of structuring your Web application.

All mobile **Web Forms pages** derive from the **MobilePage class,** which itself derives from the **ASP.NET Page class.** Therefore, every mobile **Web Forms page** is a valid MobilePage object. Each MobilePage object must contain one or more **Form controls**, which, as we've mentioned, you use to group controls into programmatically accessible objects.

A mobile **Web Forms page** can contain more that one **Form control**; however, you can't nest (or overlap) these controls. Each Form control can contain one or more mobile controls. These mobile controls can be of any type, except other Form controls or style sheets. You can include one or more **Panel controls** within a Form control; doing so will allow you to dictate the appearance of groups of controls. A Form control can contain zero or more Panel controls. Unlike Form controls, you can nest Panel controls.
For example, Panel control A can contain Panel control B, and Panel control B can contain Panel control C.

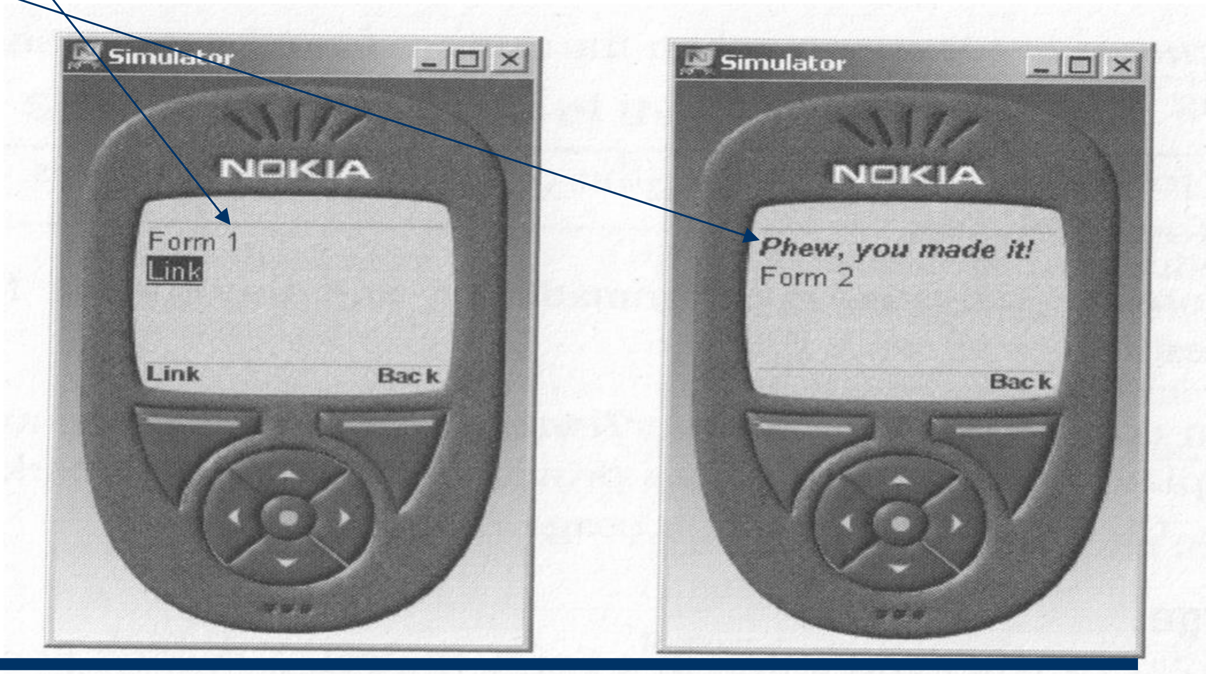

Mobile Web Forms page
Form control
Panel control
Other mobile control
Other mobile control
Other mobile control
Other mobile control
Form control
Panel control
Panel control
Other mobile control
Panel control
Other mobile control
Other mobile control

Example of an aspx file:

```
<mobile:Form
        runat="server"
        id="id"
        Font-Name="fontName"
        Font-Bold="{NotSet|False|True}"
        Font-Italic="{NotSet|False|True}"


        ForeColor="foregroundColor"
        BackColor="backgroundColor"
        Alignment="{NotSet|Left|Center|Right}"
        Visible="{True|False}"

        Action="url"
        Method="{Post|Get}"
        OnActivate="onActivateHandler"
        OnDeactivate="onDeactivateHandler"
        OnInit="onInitHandler"
        Paginate="{True|False}"
        PagerStyle-NextPageText="text"
        PagerStyle-PageLabel="text"
        PagerStyle-StyleReference="styleReference"
        Title="formTitle"> Child controls
</mobile:Form>
```

Source code for FormExample.aspx

```
<%@ Register TagPrefix="mobile"    Namespace="System.Web.UI.MobileControls"Assembly="System.Web.Mobile" %>
<%@ Page language="c#"    Inherits="System.Web.UI.MobileControls.MobilePage" %>
<mobile:Form id="Form1" runat="server">
  <mobile:Label id="Label1" runat="server">       Form 1     </mobile:Label>
  <mobile:Link id="Link1" runat="server" NavigateUrl="#Form2">       Link     </mobile:Link>



</mobile:Form> <mobile:Form id="Form2" runat="server">
   <b>       <i>Phew, you made it!</i>     </b>
   <br>
   <mobile:Label id="Label2" runat="server">       Form 2     </mobile:Label>
</mobile:Form>
```



State management in mobiles

State Management in mobiles

When you build dynamic Web applications, you usually need a mechanism to store information between client requests and server responses. Unfortunately, Hypertext Transfer Protocol (HTTP) is effectively stateless, which means you must maintain state in some other way.
In the past, developers often used cookies to track—and thus identify—a user with a session ID and to reconcile information stored on the server to that user

Following are four significant methods ASP.NET offers for preserving state:

1. Session state Allows you to maintain the variables and objects for a client over multiple requests and responses.
2. Hidden variables Allow you to persist objects between server round-trips by posting the data to the client as hidden fields.
3. View state Allows you to maintain the values of a mobile Web Forms page on the server. The runtime stores this information in an instance of the StateBag class, which itself gets stored within the session. We'll discuss later (in the section "View State,") the server then sends some information to the client.
4. Application state Allows you to maintain the variables and objects of an application over multiple requests by multiple clients.

1. Managing Session State

ASP.NET offers an updated and improved version of the Session object. This object allows you to perform the following tasks:

-Identify a user through a unique session ID.

-Store information specific to a user's session.

-Manage a session lifetime through event handler methods.

-Release session data after a specified timeout

The Session property of the System.Web.HttpApplication class (the parent class of the Global.asax page) and the Session property of the MobilePage class (the parent class of your mobile Web Forms page) both give access to the Session object.

Typically, you'll manipulate the Session object either in

1. the code-behind module of your application's Global.asax file or
2. the code-behind module of your mobile Web Forms page.

Like mobile Web Forms pages, the Global.asax file supports a code-behind module. This module follows the naming convention global.asax.extension, where extension indicates the programming language used. For example, you'd name a C# code-behind module Global.asax.cs.

Following is a fragment of code that adds a string representing the user's start time to the Session object with a key of UserStartTime from within the Global.asax file.

A way of adding items to the Session object is shown:
the use of Add method to define an entry with the key HelpAccess.

Global.asax.cs file for the SessionObjectExample project:

Constructed for every new session

```
….
namespace SessionState1
{
            public class Global : System.Web.HttpApplication
            {
                        protected void Session_Start(Object sender, EventArgs e)
                        { Session["UserStartTime"]=DateTime.Now.ToLongTimeString();
                        Boolean HelpAccess=false;
                        Session.Add("HelpAccess",HelpAccess);
                        }
            }
}
```

The Global.asax file that references the code-behind module consists of a single line containing just an @ Application directive:

```
<%@ Application Codebehind="Global.asax.cs"    Inherits=" SessionObjectExample.Global" %>
```

MobileWebForm1.aspx of project SessionObjectExample:

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
                                Assembly="System.Web.Mobile" %>
<%@ Page language="c#" Codebehind="MobileWebForm1.aspx.cs"
                                Inherits="MobileWebForm1" AutoEventWireup="true" %>

<mobile:Form id="Form1" runat="server">
                                <mobile:Label id="Label1" runat="server"/>
            <mobile:Command id="Command1" runat="server">Go To Help
                                </mobile:Command>
</mobile:Form> <mobile:Form id="Form2" runat="server">
                                <mobile:Label id="Label2" runat="server">
                                          This is a help page.    </mobile:Label>
                                <mobile:Label id="Label3" runat="server">
                                          </mobile:Label> </mobile:Form>
```

MobileWebForm1.aspx.cs of project SessionObjectExample:

```
public class MobileWebForm1 : System.Web.UI.MobileControls.MobilePage
{
        protected System.Web.UI.MobileControls.Label Label1;
        protected System.Web.UI.MobileControls.Label Label3;
        protected System.Web.UI.MobileControls.Command Command1;

        protected System.Web.UI.MobileControls.Form Form1;
        protected System.Web.UI.MobileControls.Form Form2;
        private void Page_Load(object sender, System.EventArgs e)
        {               Label1.Text = "Help accessed: ";
                        Label1.Text += Session["HelpAccess"].ToString();
                        Command1.Click += new System.EventHandler(Command1_OnClick);
        }
        private void Command1_OnClick(object sender, System.EventArgs e)
        {       //Switch to the Help form, set the flag in Session object
                        Session["HelpAccess"] = true;
                        Label3.Text = "Help accessed: ";
                        Label3.Text += Session["HelpAccess"].ToString();
                        ActiveForm = Form2;
        }
}
```

When this simple application executes, the HelpAccess flag in the Session object is initialized as false in Global.asax.cs. When the Web Form displays, the label on Form1 displays a message to show that this is the case.
When the user clicks the Command control marked Go To Help, the Command1_Click event handler in the code-behind module executes, setting the HelpAccess flag to true and setting the text of Label3 to reflect this new state.

2. Working with Cookies

Cookies provide an invaluable way for Web servers to identify wired clients such as HTML desktop browsers. However, the potential of cookies is limited with regard to applications for wireless clients. This is because many wireless devices, including some Wireless Application Protocol (WAP) and i-mode devices, don't support cookies.
If you know that your target devices support cookies or that a proxy supports them on the client's behalf, as is the case with some WAP gateways, cookies provide an excellent way to track and identify sessions.

3. Hidden Variables

Sometimes you might want to pass small amounts of information between Web pages without using session state. For example, suppose you need to collect information from a form that the user fills in. In HTML, you'd pass the information from one page to another using input tags with a type value of hidden.
The MobilePage class's HiddenVariables property provides this type of functionality. This property allows you to store variable name-value pairs, which the runtime then passes back and forth between the server and the client as hidden fields.

You should use HiddenVariables only for small amounts of information.

Example: *…*

HiddenVariables.Add(TextBoxName.ID,TextBoxName.Text);

4. View State

ASP.NET gives the user the impression that the runtime maintains pages over several server round-trips. The pages don't really exist over multiple requests and responses; instead, the runtime saves the properties of the page into a server control view state (an instance of the **StateBag class**). When the user makes a request, the runtime automatically reconstructs the page using the property values persisted in the StateBag instance.

For example, if you define a property in your code-behind class, that property isn't automatically saved and restored each time the page is torn down and then reconstructed on the next request. If you set this property in code on one request, you might want to persist this value across server round-trips.
You could add the property to the session or even persist it by using hidden variables.
However, if you use the **ViewState property** of the MobilePage to maintain the property's value, the runtime will automatically save and restore that value on your behalf.

```
…public String MyMessage              {
            get
            {         // Explicit cast to String
            return (String) ViewState["MyMessage"];
            }
            set
            {
             ViewState["MyMessage"]=value;
            }                          }
private void Command1_Click(object sender, System.EventArgs e)
{       // Consume the persisted property.
            Label1.Text=this.MyMessage;
}
```

When using the session with view state, you have two important considerations.
First, sessions can expire, which means you can lose your view state information.
The number of minutes allowed to elapse before a response is received from a client is set by the timeout attribute of the sessionState element in the application's Web.config file;
20 minutes is the default.

```
<sessionState
            mode="inProc"
            cookieless="true"
            timeout="20"          />
```

Second, the page displayed on the client and the current state of the session information held on the server can fall out of sync. This can occur when a user uses a Back feature on the browser to return to a page viewed previously—for instance, by pressing a Back button. For example, imagine that a user goes to the first page of an application and then clicks a link to go to the second page. If the user then navigates backward to the first page, the user views the first page while the server holds session data for the application's second page. The Mobile Internet Toolkit overcomes this issue by maintaining a small history of view state information in the user's session.
You can configure the size of the view state history. The default history size is 6. To change the history size, use the sessionStateHistorySize attribute of the mobileControls element within the Web.config file, as the following code shows:

```
<configuration>
  <system.web>
            <mobileControls sessionStateHistorySize="10"/>
   <system.web>
</configuration>
```

5. Application State

In ASP.NET, an application is the total of all the files that the runtime can invoke or run within the scope of a virtual directory and all its subdirectories. At times, you might want to instantiate variables and objects that have scope at an application level rather than at a session level.
The HttpApplicationState class allows you to do this. The Application object is the generic term for the instance of this class for your application, which is exposed through the Application property of the System.Web.HttpApplication class (the parent class of the Global.asax page) and the Application property of the MobilePage class (the parent class of your mobile Web Forms page).

Using Application State in Global.asax

You define information that relates to application state in the Global.asx file, which always resides at the root of a virtual directory. For the moment, you'll use Global.asax to implement event handlers associated with application state, but remember that you can also use this file to store other information such as session state, as described earlier.
You define application state data within the code-behind module of Global.asax by writing code for the two event handler methods":

Application_Start
and
Application_End.

Things to Consider When Using Application State

At times, you might wonder whether to use session state or application state.

- First of all, information stored in application state is memory hungry.
In other words, the application holds all application state information in memory and doesn't release the memory, even when a user exits an application.

- Second, all threads in a multithread application can access application data simultaneously, since ASP.NET doesn't automatically lock resources.