Programming for mobiles

theory;
different OS;
frameworks;
good practices.

prof. O. Nakov, Ph.D

History

In 2007 Apple introduced the iPhone . No SDK available. No App Store
Latter in 2008 Apple launched the App Store with more than 500 available apps
In a 3 days more than 10 million downloads occurred

In 2088 Google announced Android Market. By summer 2010 there were 80 000 apps and 1 billion downloads.
In the same time Apple Store had 225 000 apps and 5 billion downloads

In October 2010 MS announced Windows Phone Marketplace and Windows Phone SDK.
By July 2011 there were 26 000 apps available.

In July 2011 mobile industry is estimated to represent 2% of the world's product

The challenge is software technology standardization:
- .Net and C#
- Java ..
- Mono technologies (MonoTouch and Mono for Android), Cordova (PhoneGap)

are needed to deliver functionality to multiple platforms with little or no modifications.

Introducing Windows Phone Programming

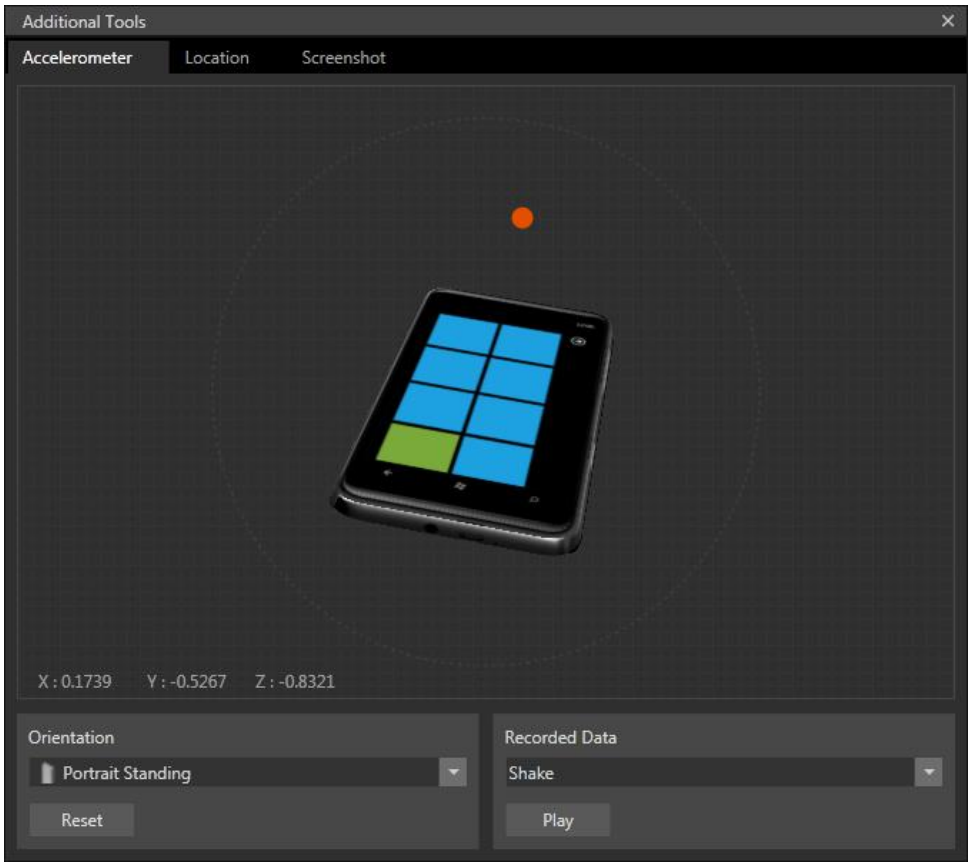
- Writing programs for Windows Phone is the same as writing for any .NET platform
- Edit, compile and debug within Visual Studio
  - But you need to remember you are writing for a platform a bit more constrained than a PC
- You can incorporate .third party NET libraries (assemblies) into your applications

Windows Phone Emulator

- The emulator runs on your PC
- It contains exactly the same code as the real phone, but compiled for the Windows PC
- It lets you see what your programs look like on a device
  - It does **not** show you what the application performance will be like on a real device

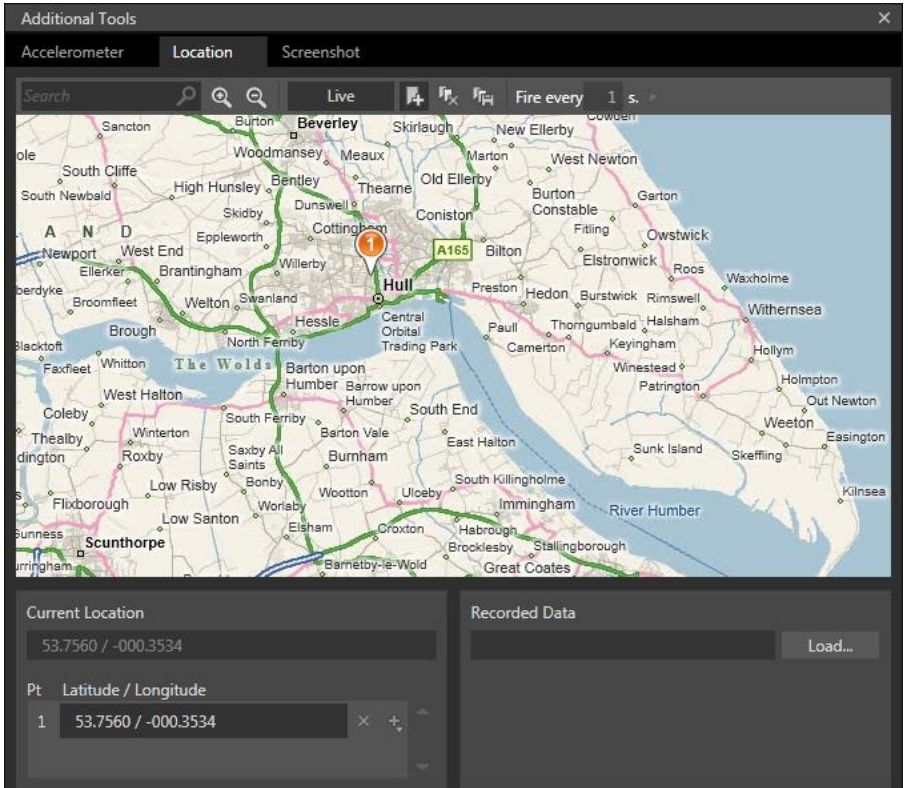
Orientation Emulator

- The emulator allows you to “move” a 3D phone
- The signals sent to the sensors in the emulator match the viewed position
- You can record and replay movements



Location Emulation

- You can select on the map the “location” of the emulator
- You can also record and playback particular routes



Using Windows Phone from software

- Windows Phone provides a library of “Launchers” and “Choosers” that your programs can invoke
  - Launchers start other tasks in the phone
    - Start a phone call
    - Post a status message to a social network
  - Choosers allow the user to select an option and then restart your program so it can use the returned value
    - Select a contact from the Address Book

Network Connectivity

- A Windows Phone will be able to use the 3G phone network and WIFI to connect to the Internet
- Your programs will be able to interact with servers, call web services
- In the present version of the operating system there is no support for direct socket connections

Silverlight and XNA applications

- You can build two kinds of applications for a Windows Phone
- Silverlight
  - Business applications and simple casual games
- XNA
  - 2D and 3D games with hardware accelerated graphics
- Combined Silverlight with XNA graphics

Silverlight Applications

- This is not a very good looking Silverlight application
  - But it does show that you can build displays for user applications using Silverlight
- There are lots of custom display components for the phone that you can use

XNA Applications

- XNA is a game development environment for Windows PC, Xbox and now Windows Phone
- Existing XNA games are very easy to move onto the phone
- The phone provides 3D graphics support for games

Creating an Application

- You select the type of your application (XNA or Silverlight) when you create the new project in Visual Studio
- You can create a Silverlight application that includes an XNA game on one page
- There is no technical reason why you could not create a Silverlight game or an XNA business application

Development Tools

- The Development Tools for the platform are a free download
- Tools are chosen for the version of Visual Studio and the Windows Phone emulator
  - These will integrate into an existing Visual Studio installation
- You can also obtain free versions of the Expression Blend user interface design tools for Silverlight

Windows Phone Marketplace

- You can develop and test your application on the emulator for free
- To sell your application or deploy it to a real device you must register as a developer
- This costs $99 per year ($1 for students)
- Students can register as developers for free via Microsoft DreamSpark

Marketplace Rules

- In a year of your subscription you can publish any number of applications for sale and up to 100 free ones
  - To distribute more free applications costs $20 per application
- When you sell an application you get 70% of the price you charge
- You can distribute “demo” and “time trialled” versions of your application

Marketplace approval

- A program submitted for sale via marketplace is submitted to an approvals process
- This includes checks for matters of taste and decency, along with proper application behaviour
- If the program fails the process you will be given a report and can re-submit the application

Review

- Windows Phone programs are developed in the same way as any other .NET application
- The Windows Phone emulator does not emulate the speed of the platform
- Programs can make use of phone functions
- Windows Phone programs either Silverlight (business) or XNA (game) based
- The development tools are all free
- You need to join the marketplace to sell apps

Design practices for building a better mobile applications

- Mobile devices are the most common way to access the Web within last five years.
- in America, 25 percent of mobile Web users say they “never” or “infrequently” access the Web using a traditional PC

Mobile Programming is Different

Every mobile browser supports some form of HTML. Many, especially on high-end devices such as iPhones and Windows Phone 7, support the latest HTML5, CSS and JavaScript standards and render pixel-perfect copies of what you’d see in a traditional PC browser.

Your cheapest option for supporting mobile browsers, then, **is to do nothing**.

Choosing this option leads to a **very poor mobile browsing experience**, for several reasons:

1. screens are small.

The most obvious difference between a PC and a mobile phone is display size. The average size of a PC monitor is 21 inches, with a display resolution no less than 1024 × 768.
The typical smart phone has a display of 3.9 inches and a resolution less than 800 × 480. The mobile phone doesn't give you much space to work with. You need to use less text and better icons.
The figure bellow illustrates an effective way of presenting clear icons that represent key functions.

Some mobile browsers, such as ***Opera Mini***, handle desktop-width pages by dynamically reformatting the page layout and styles. The resulting appearance is rarely what your designer had in mind.

Other mobile browsers, such as ***Safari*** for iPhone or ***Internet Explorer*** for Windows Phone 7, render desktop-width pages and then force the user to zoom in and out and pan around to read the text.

If you rely on icons to convey your features, functions and contents, your icons need to be **more concrete and less complex** than for a PC, and they must accurately represent the concept you're trying to express.

Your **text should be as legible and readable** as possible. Legibility refers to how easy characters and words can be identified. To make your text more legible, use a font size of at least 10 points and turn up the contrast.

Here's what you need to remember about display size:

- ***You don't have much room to work with, so minimize the number of screens and options, and minimize the amount of text.***
- ***The text you use should be legible and readable.***
- ***Replace text with icons when you can.***
- ***Your icons should look as realistic as possible.***
- ***The visuals should match the tasks, and the design should be straightforward.***

2. Input is different

The standard PC has several ways of inputting, but basically a keyboard and a mouse are used.
A mobile device relies mostly on **fingers**, whether they are used to type on the screen keyboard, **tap icons or buttons, or make gestures, such as swiping or pinching**.

To compensate for these drawbacks, you can provide **sound or vibration for key presses** and minimize text entry.
Fingers comes in all sizes, so make your **buttons large enough** (35 pixels square) and visually separated from other buttons or objects to reduce error.

Find innovative ways to input data, such as **using photographs.**

You should also take advantage of the hardware. **Windows Phone has three dedicated buttons**
Back,
Start
Search

and you can use them to reduce clutter on the interface and to minimize error.

3. Context: Where Phone apps are used

PCs are used mostly in homes and offices—fairly predictable and stable places, with good lighting, limited distractions and a relatively focused user.

When developing for mobile users, throw out everything you know about PC users. Mobile users are a different . Mobile device users (for the most part) are on the move, and many are in a rush.
They have a limited amount of time and have come to your application for a specific reason. They come with questions that need to be answered.
If your application is to succeed, it must help users get those answers quickly. If users are consistently helped by your application, they will spend more time using it.

Environmental issues that rarely occur at the desktop often appear in a mobile environment. Distractions such as noise can draw the user's attention away from the device, or at least interfere with the detection of sounds from your application. Sun glare can obliterate the screen and make it difficult even to see your application much less use it. The overload of information can make it difficult for users to concentrate on your application, which can affect their cognitive processing.

One final note: Developers who simply miniaturize the PC experience for the mobile device are missing the point. We need to focus our attention on giving users what they want most from our applications— simply but elegantly.

4. Mobile data networks are often slow

Don’t assume that your visitors have the same bandwidth as fixed-line broadband users.
They may even be paying by the megabyte, so heavyweight sites won’t be popular.

There are two main aspects when working with mobile browsers:

1. Detecting which kind of device a given visitor is using.

ASP.NET has built-in support for browser detection. In the next section, we'll examine this mechanism.

2. Producing output that works well on the detected device.

we'll describe technical means to produce different outputs for different devices, but it's still up to you to design and implement different layouts and user workflows for mobiles.

Browser Detection

You can find out whether or not a visitor is using a mobile browser using (ASP.NET) the **Request.Browser.IsMobileDevice** Boolean property.

ASP.NET determines what kind of browser is making a request and what capabilities that browser has (screen size, JavaScript support and so on) by comparing the incoming request userAgent header string against a series of regular expressions in XML files that describe common browsers.

The information about corresponding device capabilities is stored in a set of .browser files in the folder :

C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\Browsers

(or your installation's equivalent).

For example, the standard iphone.browser file includes the code shown:

```
<browsers>
<!-- Mozilla/5.0 (iPhone; U; CPU like Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko) Version/3.0
                    Mobile/1A543a Safari/419.3 -->
  <gateway id="IPhone" parentID="Safari">
      <identification>
        <userAgent match="iPhone" />
      </identification>

      <capabilities>
        <capability name="mobileDeviceModel" value="IPhone" />
        <capability name="mobileDeviceManufacturer" value="Apple" />
        <capability name="isMobileDevice" value="true" />
        <capability name="canInitiateVoiceCall" value="true" />
      </capabilities>
  </gateway> ...
</browsers>
```

The following element defines the expression to be matched against incoming userAgent header strings:

<userAgent match="iPhone" />

Once the system finds a matching userAgent expression, the remainder of the XML data specifies the type and capabilities of that device.

(Unfortunately, this does not include common modern browsers such as Opera Mobile or the default browser for Google Android. Request.Browser.IsMobileDevice will incorrectly be set to false)

How to Enhance Browser Detection

You have two main options :

1.You can supply your own .browser files to represent newer devices.
2.You can use a third-party browser-detection library.

To take the first option, right-click on your project's name in the Visual Studio Solution Explorer and choose Add | Add ASP.NET Folder | App_Browsers. You can then add .browser files to that folder;

If you don't want to be responsible for tracking all of the hundreds of newly released mobile browsers and keeping your .browser files up-to-date, you can take the second option and use a third-party browser-detection library.

Currently, the one is 51degrees.MobiFoundation, an open source Library.
This library does not use .browser files directly.

The easiest way to install 51degrees.Mobi Foundation into either Web Forms or MVC projects is by using the NuGet package manager.

If you're running ASP.NET MVC 3+, you already have NuGet. If not - use the VC Extension Manager (it's on the Tools menu) to install NuGet. Go to Tools | Library Package Manager | Package Manager and issue the command:

Install-Package 51Degrees.mobi

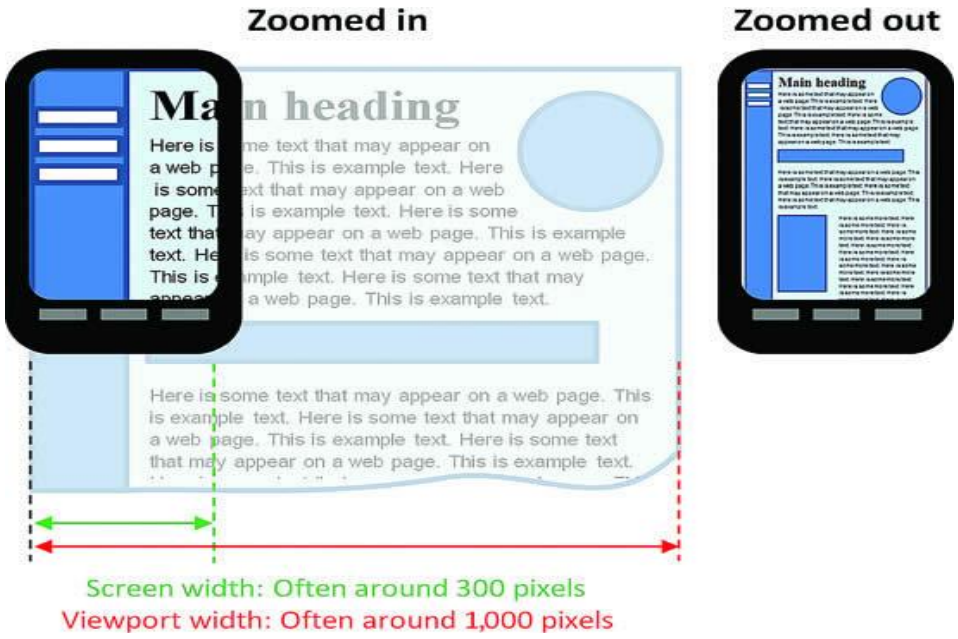
Styling for Mobiles

Now that you have an idea of how to detect mobile browsers reliably, We'll show a key way to control how pages are rendered by mobile browsers.

Many mobile browsers, including Safari for iOS and Internet Explorer for Windows Phone 7, try to make rendered pages look just as they do on a desktop browser. They know that most pages are designed for screens around 1,000 pixels wide and the designer has most likely not accounted for much smaller widths.

To solve this, they typically render the page onto a virtual canvas known as a "viewport," usually around 1,000 virtual pixels wide. The browser can then scale the visual display of that virtual canvas arbitrarily, allowing the user to zoom in and out and pan around.

This arrangement is illustrated:



Controlling the Viewport Width

If you've actually designed pages for the small screen, you'll want your pages to be laid out on a viewport that's the same width as the actual screen, so that it neatly fits horizontally with no zooming required.

Many of the most popular mobile browsers support a nonstandard "viewport" meta tag that lets you control the width of the virtual viewport. For example, if you add the following to your page's <head> section, the browser will lay out the page on a viewport 320 pixels wide:

```
<meta name="viewport" content="width=320"/>
```

This is usually a much better fit for mobile phones.

Keep in mind that some mobile devices have screens with much higher horizontal resolution. For example, the iPhone 4 has 640 physical pixels per row. However, it still makes sense to use a virtual viewport of around 320 pixels; otherwise, the resulting text will be too small to read without zooming in.

If you want, you can let the virtual viewport vary in size according to the device being used, using the following syntax:

```
<meta name="viewport" content="width=device-width"/>
```

Note that some mobile devices won't give you a literal device width. They interpret "device-width" as meaning "the virtual viewport width that the manufacturer thinks gives the most pleasing result." So, for example, iPhone 4 defines device-width as 320 pixels, despite its higher physical resolution.

Markup Recommendations

Whenever you're designing pages for mobile browsers:

1. Use the **viewport meta tag** to make the viewport fit the horizontal width of the screen.

2. Adjust your **page layouts, CSS styles** for this narrow width. If visitors don't need to zoom or scroll horizontally, your page feels more like a native application designed for their device—a far better experience.

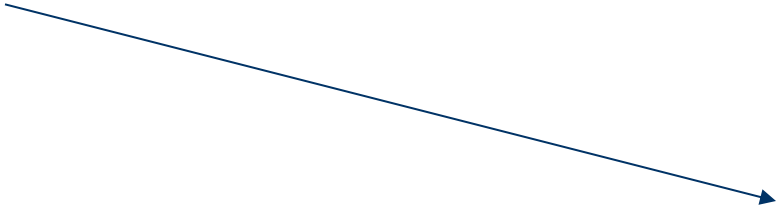
3. Make sure your **links and buttons** are large enough to be tapped imprecisely. Fingertips are much bigger than the tip of a mouse pointer.

4. **Minimize bandwidth requirements** by not using very high-resolution images or massive JavaScript files

Architectural Options

You've seen how to detect mobile browsers, and some recommendations for markup that better suits them. Now we'll describe three straightforward options for structuring your application to produce different output for different browser types:

1. **Showing and hiding sections of markup according to browser type.**
2. **Switching master pages according to browser type.**
3. **Presenting entirely different content according to browser type.**



1. Showing and Hiding Markup

If you only need to include or exclude meta tags and CSS file references according to browser type, this is extremely simple. For example, in a Web Forms master page, you can add an “if” statement inside your <head> section:

```
<head runat="server">
  <title>My site</title>
  <link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />

  <% if (Request.Browser.IsMobileDevice)
     { %>
    <meta name="viewport" content="width=device-width"/>
    <link href="~/Styles/MobileSite.css" rel="stylesheet" type="text/css" />
       <% }
    %>
</head>
```

For many sites this simple technique won’t be sufficient, but there are two alternatives:

2. switching the master page or
3. presenting different content.

2. Switching Master Pages

You may be able to keep your existing content pages unchanged, and merely adapt the layout for the small screen using a different master page or layout.

For example, if you're building a Web Forms application,
you could define a standard page base class that switches its master page dynamically:

```
public class PageBase : Page
    {
      protected override void OnPreInit(EventArgs e)
      {
        if (Request.Browser.IsMobileDevice)
            MasterPageFile = "~/Mobile.Master";
      }
    }
```

Then, for any page whose layout should vary by device type, set its codebehind class to inherit from PageBase instead of the usual System.Web.UI.Page.

You must then create a master page at /Mobile.Master whose layout and CSS styling are optimized for mobile devicess.

3. Presenting Different Content

For some applications, you won't be able to adapt your desktop pages to suit mobile devices merely using different CSS or master pages and layouts because:

Your business requirements might be too demanding. I
You may need to display different (perhaps less) information to mobile devices, and possibly guide the user through different workflows. For example, your user-registration process may have fewer steps and collect less information for mobile visitors.

You may be working with legacy code that's not amenable to such change. For example, your existing markup may contain hardcoded element sizes and styles. Modifying this using CSS or a different master page might be impossible, or might just make things more complicated and less maintainable.

In either case, the ultimate solution is to use entirely separate logic and markup for different device types. The drawback is that you then have two versions to maintain, but the key benefit is that the behavior of the two can vary independently in any way you want. For Web Forms developers, the implementation is usually a set of mobile-specific ASPX pages, and for MVC developers, it usually means creating a new area for mobile-specific controllers and views. Either way, you'll need some logic to redirect incoming visitors to the correct page depending on their device type.