Programming for mobiles

theory;
different OS;
frameworks;
good practices.

prof. O. Nakov, Ph.D

What you need to be able
to write mobile apps for different platforms?

- **.NET (C#) for example & Visual Studio;**
- **For iOS you need MonoDevelop IDE (free download) + MacBook or any Apple computer to compile and run;**
- **Android SDK from Google, and Windows Phone SDK from MS (free downloads);**
- **To install MonoTouch and Mono for Android from Xamarin (trial version or purchase) in case to develop cross-OS apps. In the trial version only iOS simulator and Android emulator are available. A license is needed to deploy an application to a device.**

Choosing the right architecture

- Always in the network;
- Storage and processor limitations . Servers, XML mediator, SOAP transfer

- Securing data on the device: if available – Lightweight Directory Access Protocol (LDAP) Username+password, data encryption; data destruction

- Building scalable applications
- Writing extendible modules: for devices (barcode, camera, geo-location etc.)

- Choosing the right software architecture:
  - Native application
  - Web application (HTML 5; CSS)
  - Application for different platforms: for iOS; Android; Windows Phone

Setting up the development environment

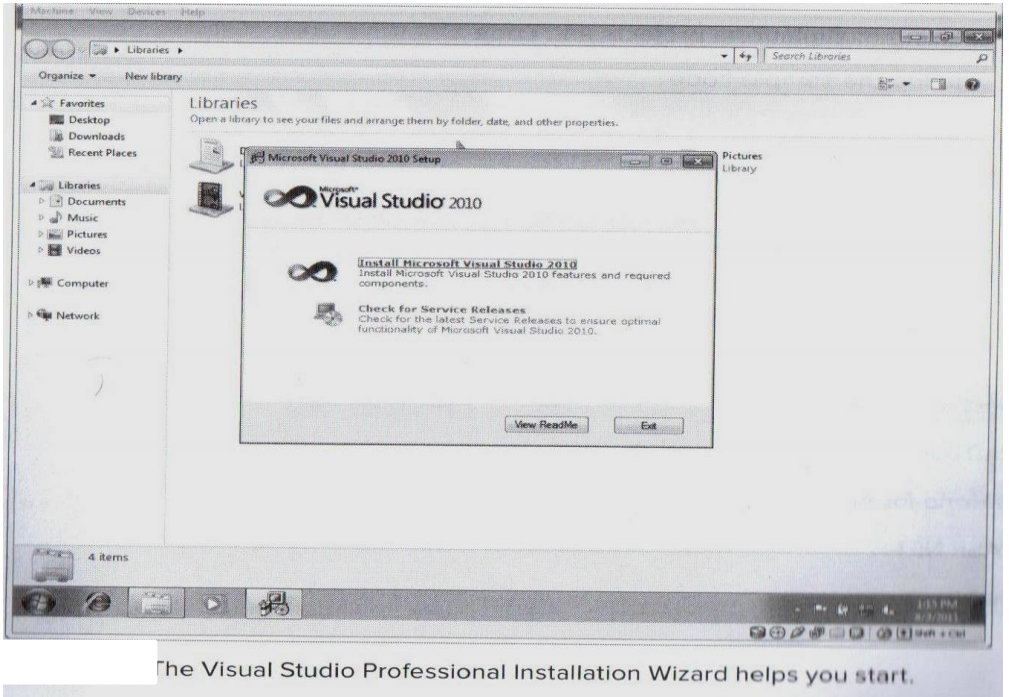
OS/ product	Mac OSX	Windows
iOS/ MonoTouch	x	
Android/Mono for Android	x	x
Webkit / ASP.NET		x
Windows Phone		x

A. Installing development tools

1. Installing MS Visual Studio (2010 up) for:

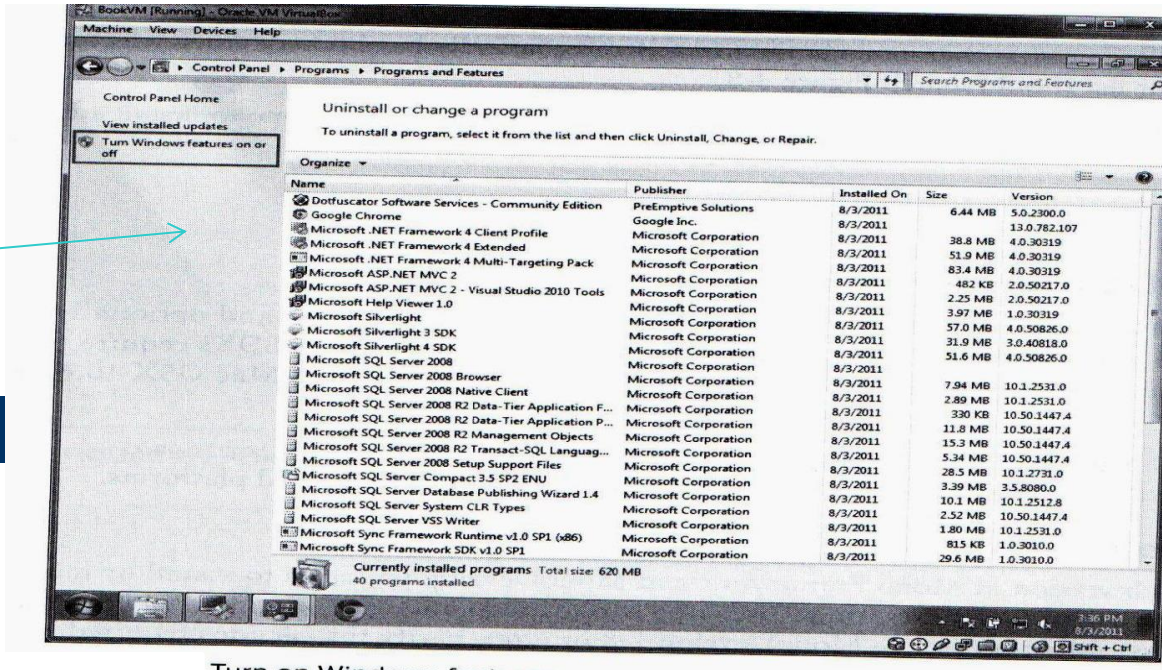
   Windows Phone 7;
   Mobile web (WebKit) ;
   Console samples ;
   apps for Android also ,

   but Mono for Android is better.

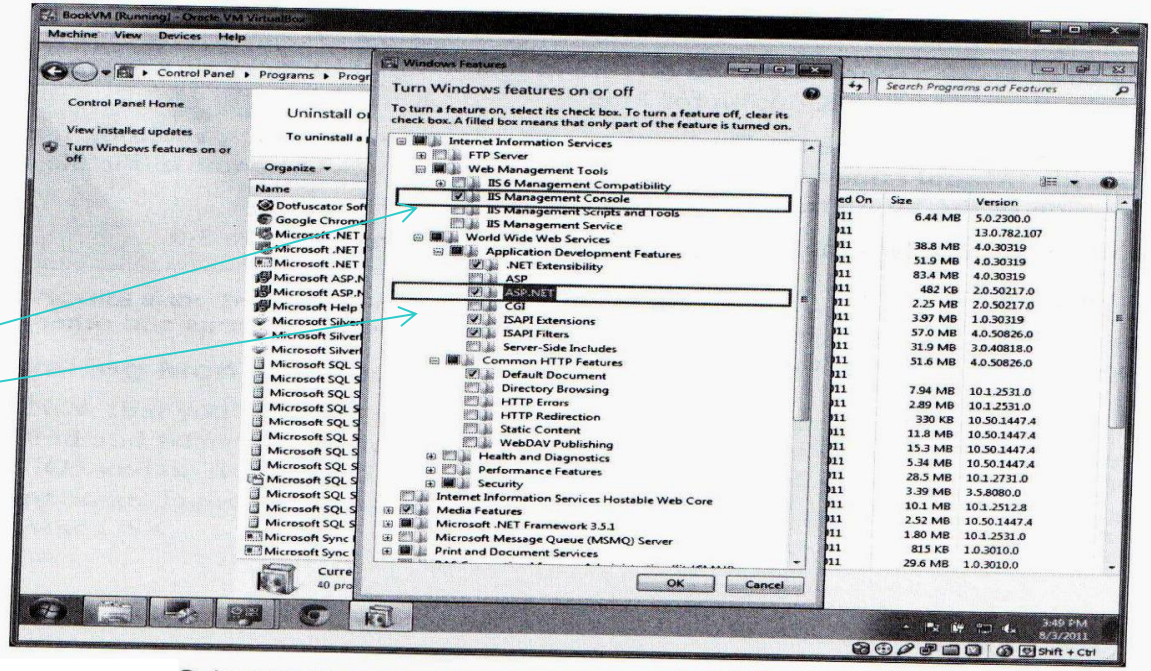


2. Download:

- **latest Service Pack for 2010 +**
- **Install IIS to be able to develop WebKit samples**



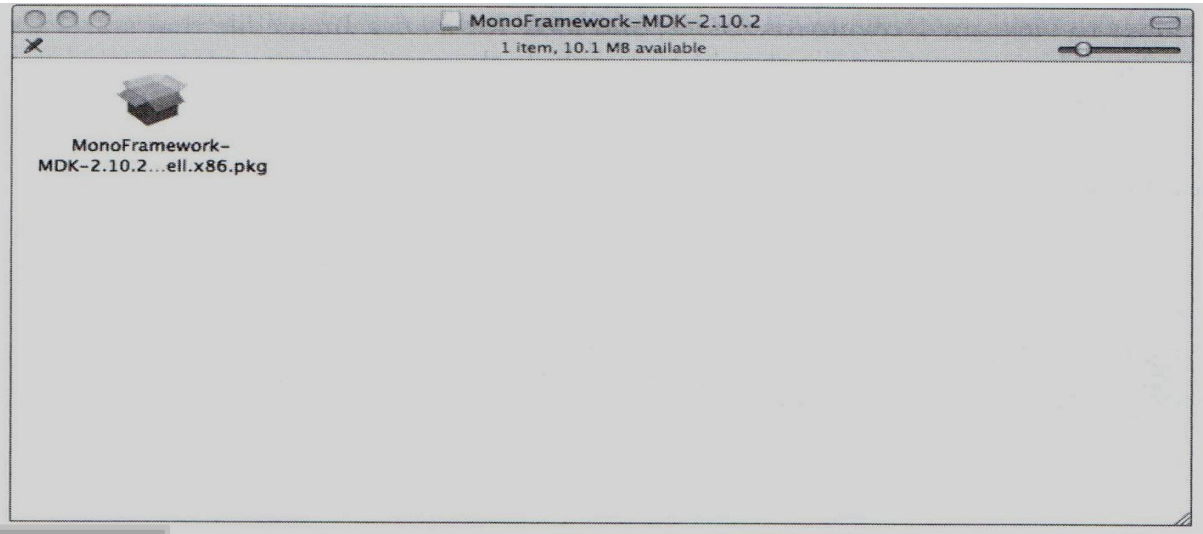
Turn on Windows features.



Select IIS Management Console and ASP.NET to continue the installation

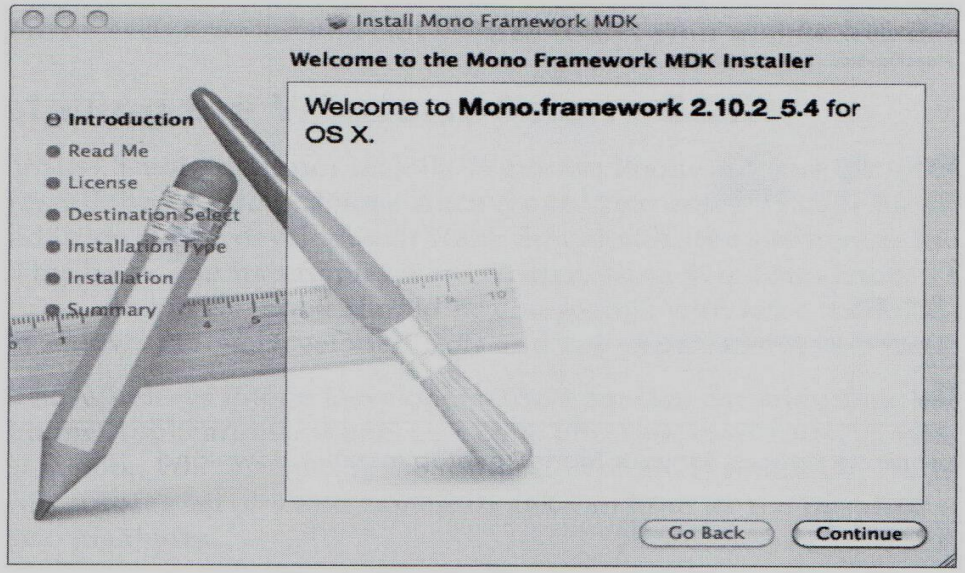
Installing MonoDevelop for Mac

- iOS SDK is required and Mac OSX 10.6 or later OS
- Download the latest version of Mono framework.

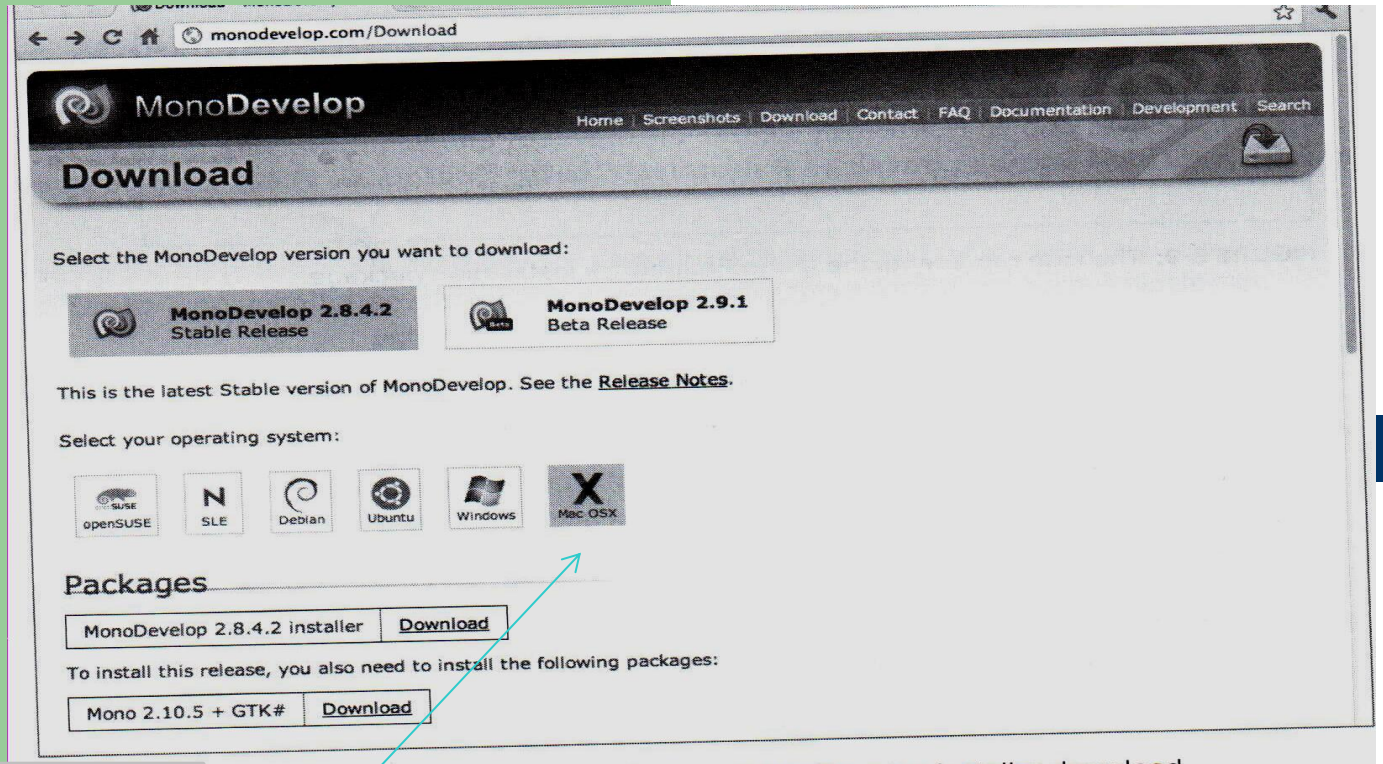


The icon represents the Mono Framework installation package.

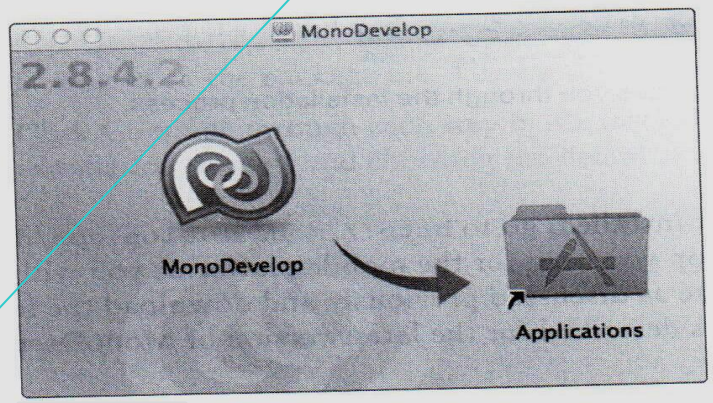
- You have the Framework. Have to install **MonoDevelop** for mobile platforms **(iOS and/or Android)**



: The Mono Framework Installer guides you through the installation process.



Select the appropriate options for your MonoDevelop Installer download.



Copy MonoDevelop to Applications.

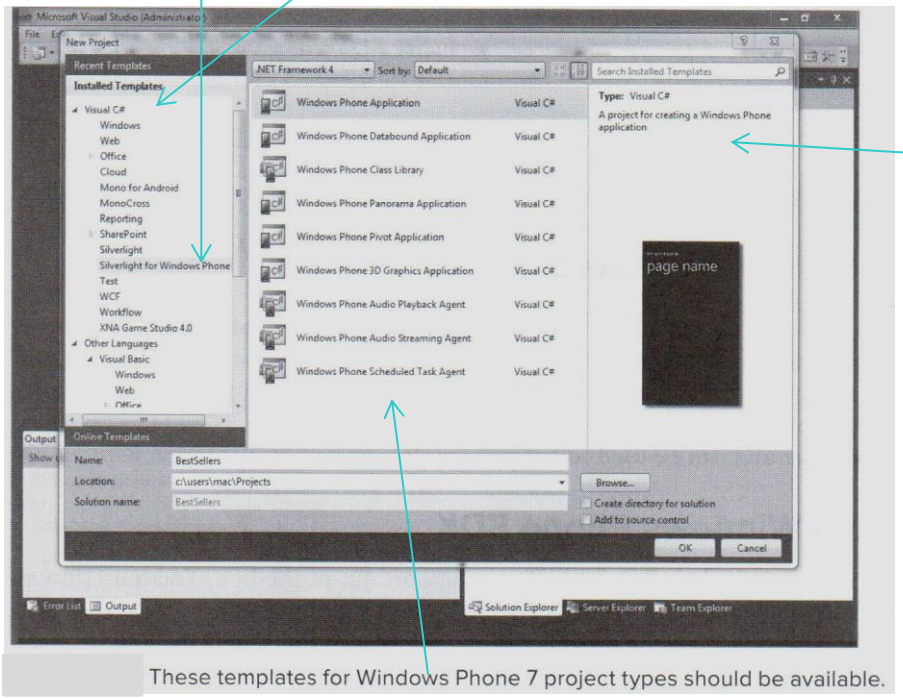
After download, start the installation and copy MonoDevelop icon into the Application folder

After having installed development tools, install the device frameworks (SDK):

B. Installing device frameworks (SDK + emulators and simulators)

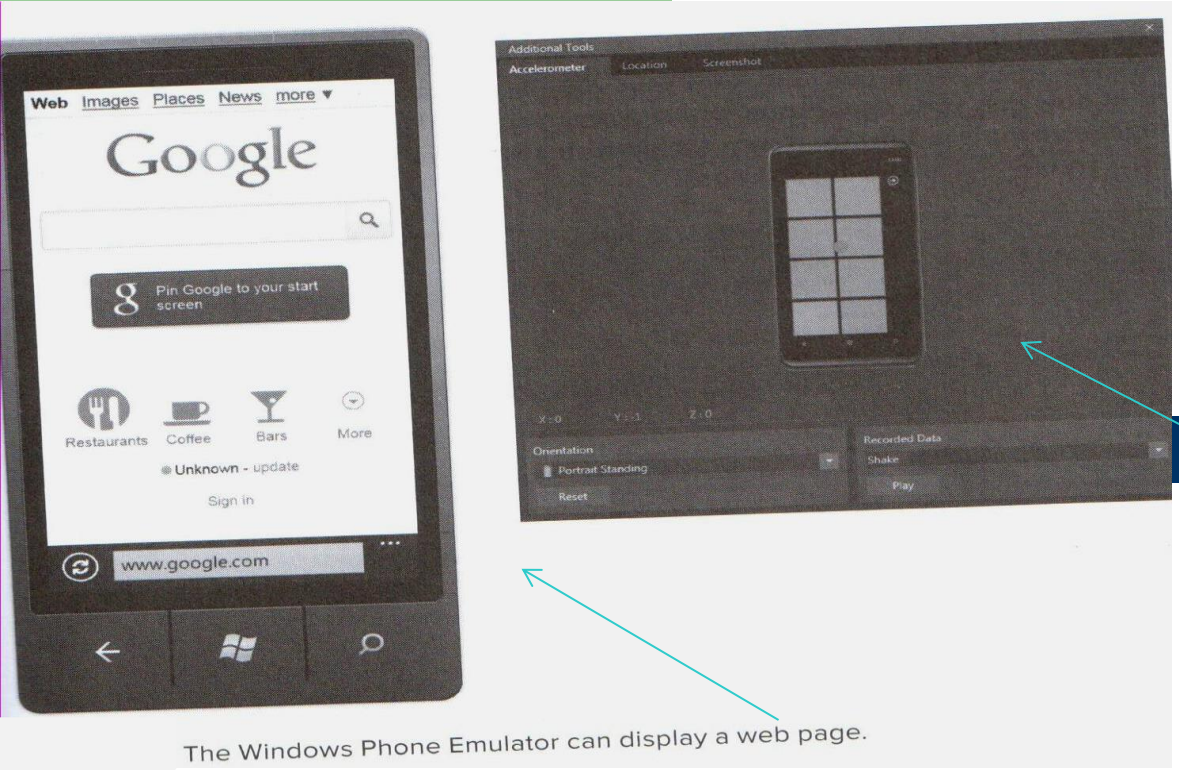
1. **Installing the Windows Phone SDK**: by downloads, free. When completed, start Visual Studio →New → Project → **Visual C#**

and you will find **Silverlight for Windows Phone:**



These templates for Windows Phone 7 project types should be available.

- You can start also Windows Phone Emulator for testing websites without starting VStudio 2010. An example of the emulator, displaying web page on a phone is shown:



The Windows Phone Emulator can display a web page.

Windows Start button;
All programs;
Windows Phone Developer tools;
Windows Phone Emulator;
Run the emulator

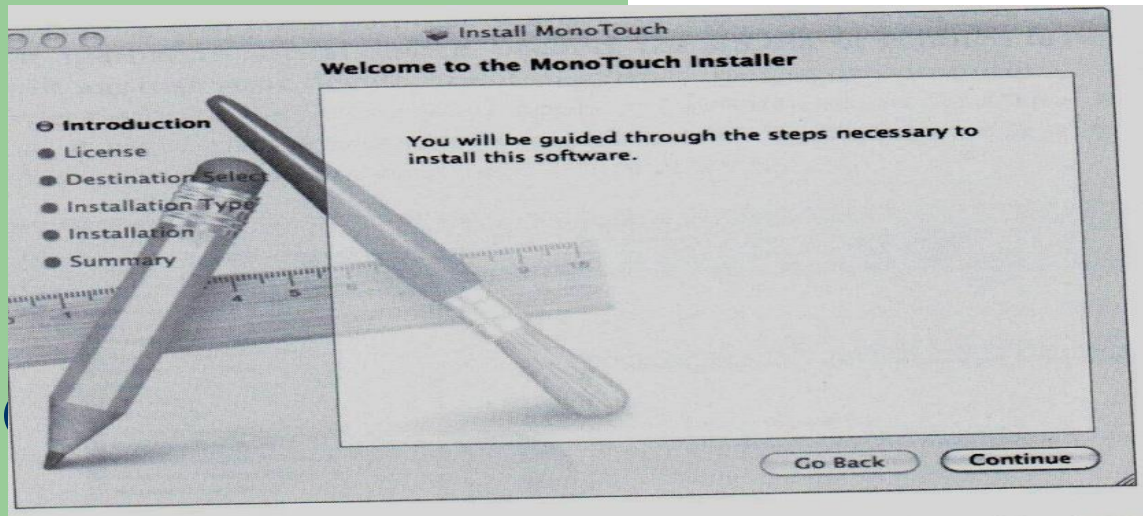
Emulation for accelerometer

2. Preparing for iOS development (iPod/iPhone/iPad)

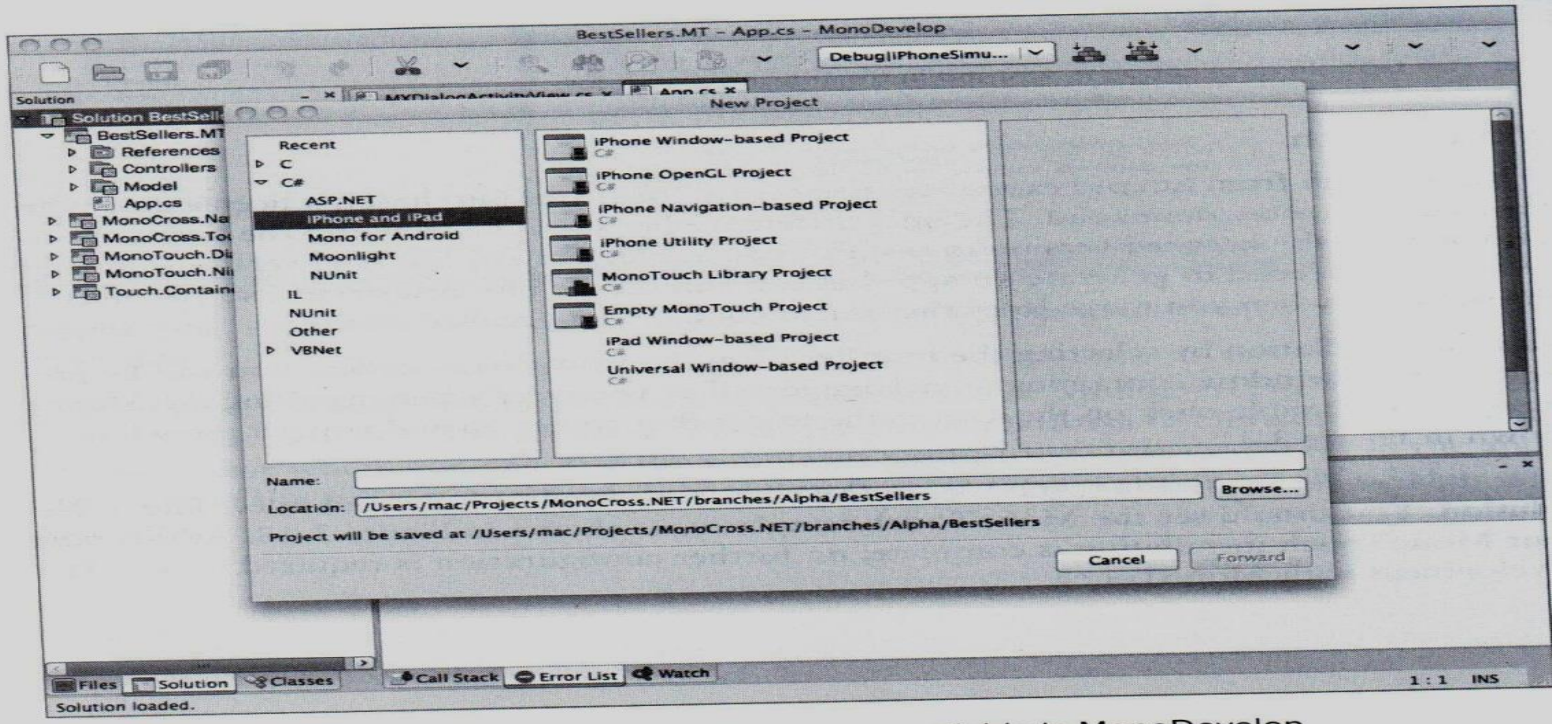
- you need iOS SDK. MonoTouch uses them to generate code from compiled .NET assemblies.
- Download Xcode developper tools + iOS SDK. All SDKs are bundled within Xcode development environment and can be downloaded when having Apple ID

You have here a simulator not an emulator (the code generated for a simulator is different from the code for target device. If emulator – the code is identical)

- Installing MonoTouch. It exists 2 versions: trial and licensed. In the trial is not included the software needed to generate an app to run on the physical device:



The MonoTouch installation application will provide steps to a successful installation.

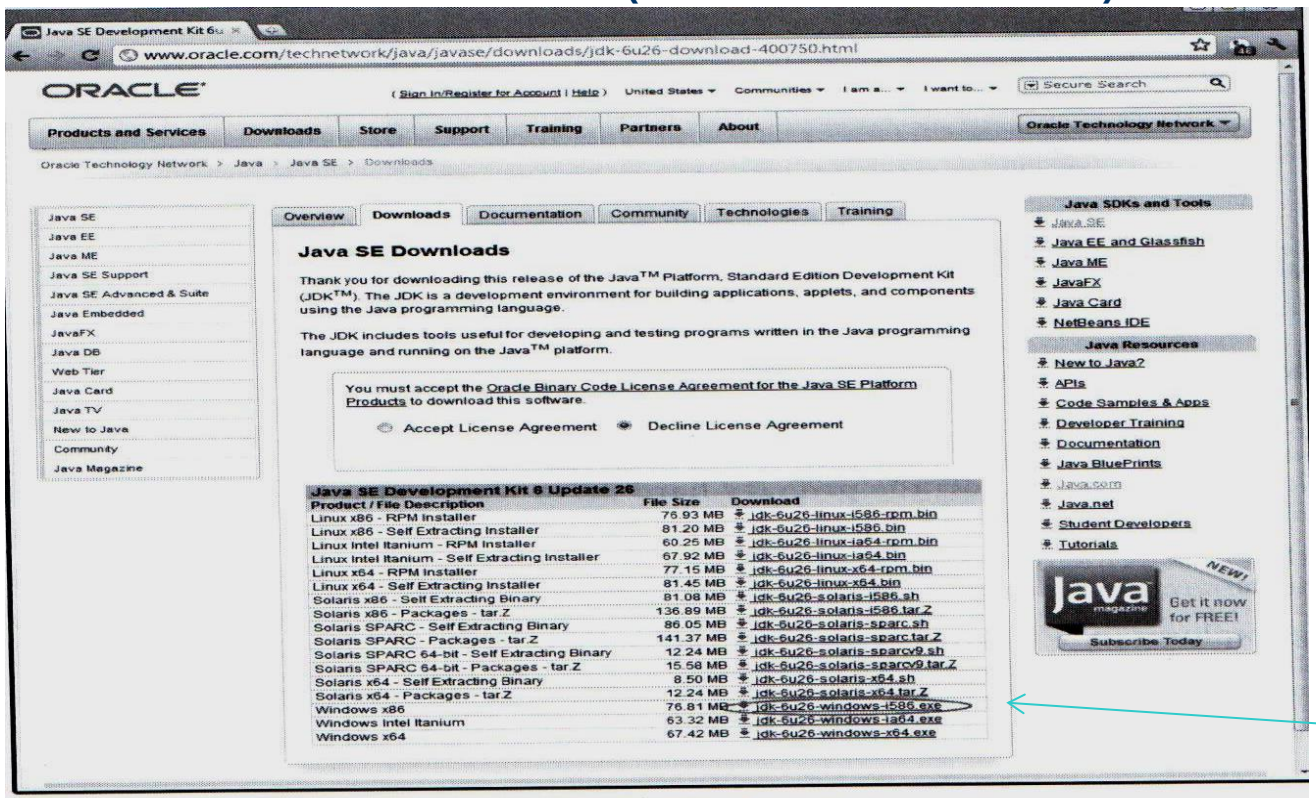


Your MonoTouch project templates are available in MonoDevelop.

3. Preparing for Android development

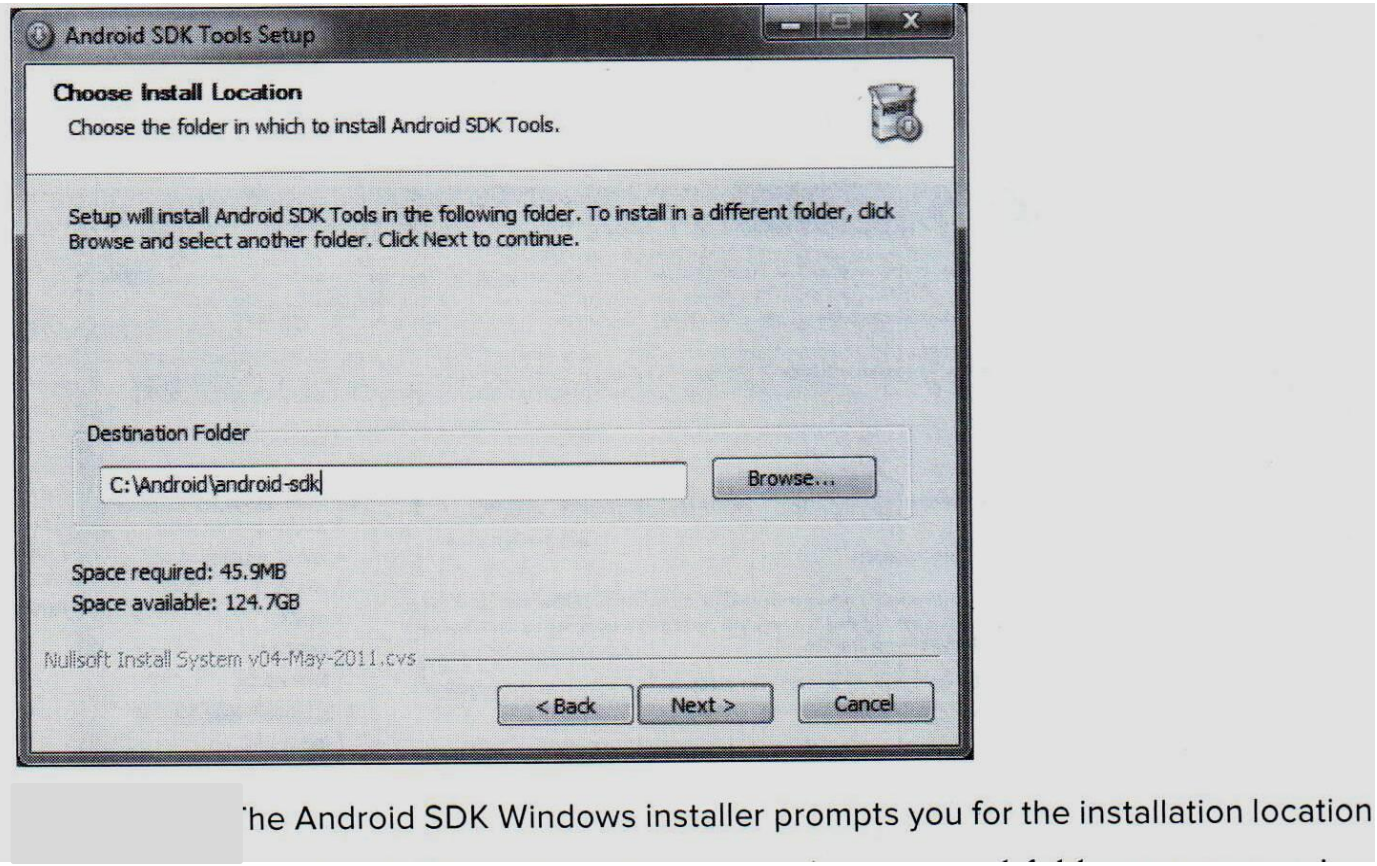
- Installation for both Windows and Mac OSX
- Tools are Java based

- Installing Java JDK:
  - For Mac OSX JDK is included natively
  - For Windows must be downloaded (32-bit version of Java JDK) from www.oracle.com



Select the correct version of the Java JDK.

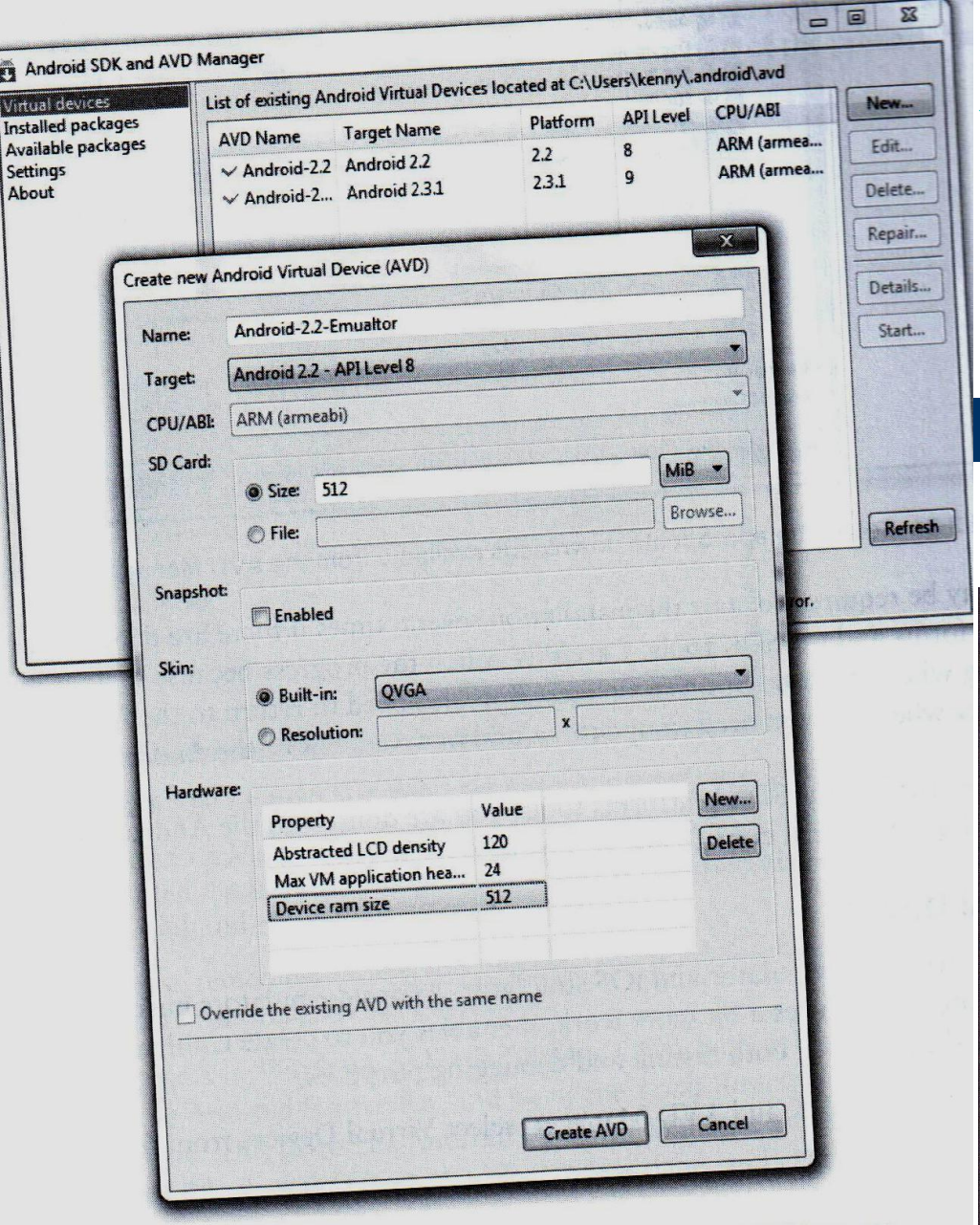
- Installing the Android SDK: must be downloaded – the suitable version (2.2; 3.x; 4.x)



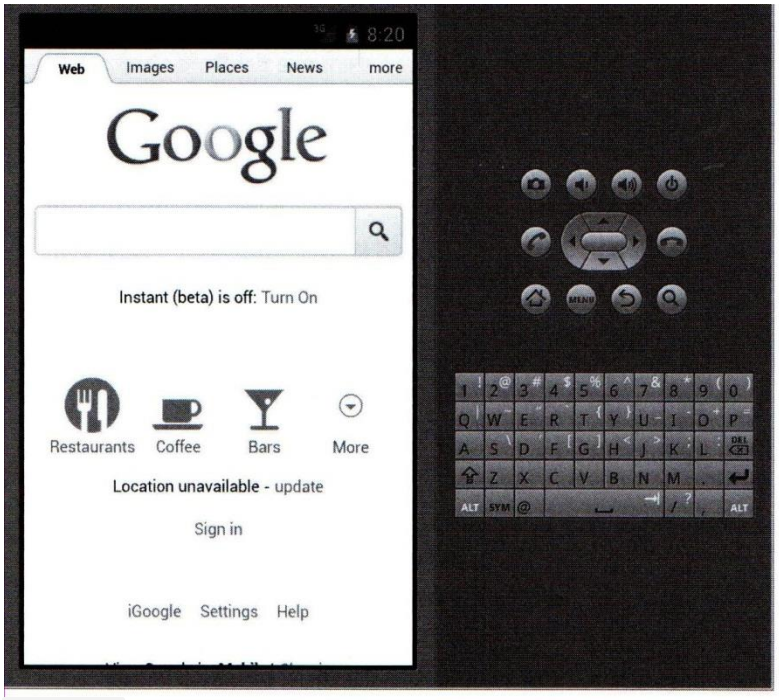
he Android SDK Windows installer prompts you for the installation location

- Configure the emulator – **add a virtual device**. Must be done explicitly!
- from the Android SDK select Virtual Devices:

- Create new AVD
- Start the VD
- You are able to use the emulator for example to run a browser:



You can create an Emulator Image from the AVD Manager.

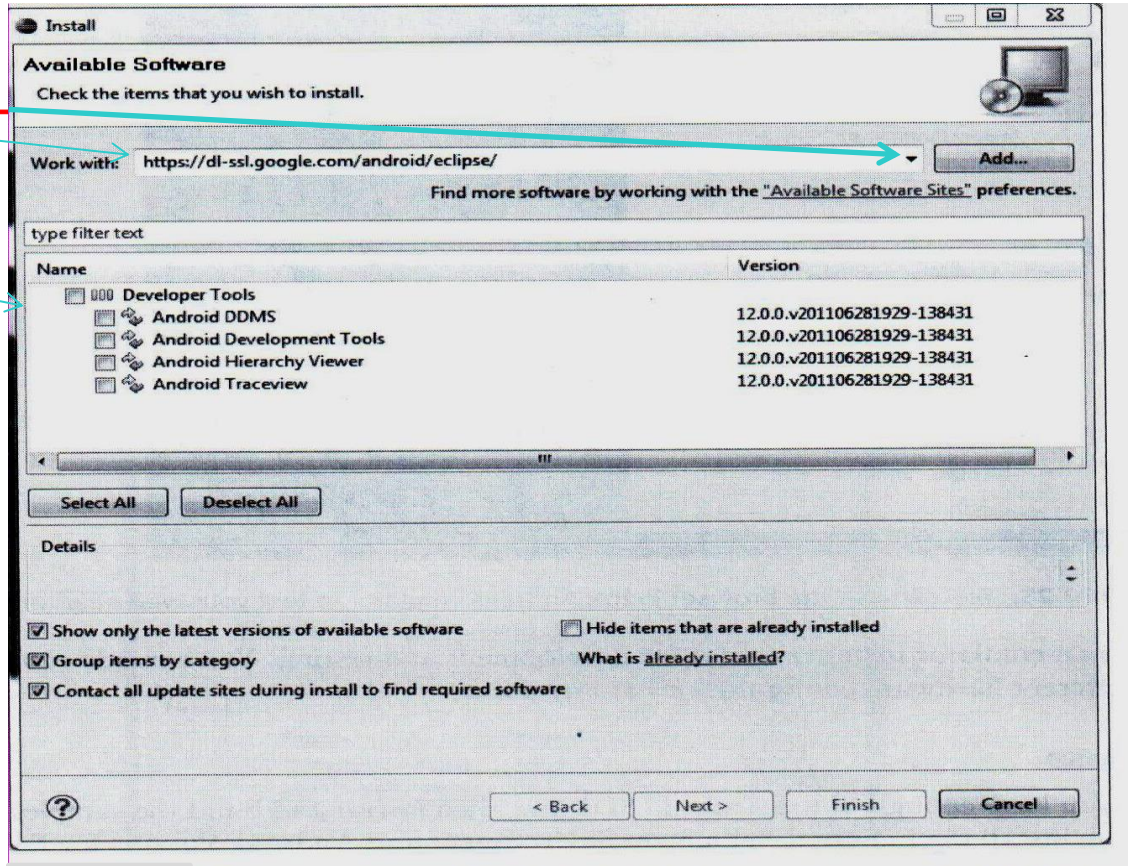


: You can use the browser in the Android emulator to test your web solutions.

4. Installing Eclipse

- it's preferable to install 1.**Eclipse development** and 2. **Android plug-ins** (installed into the Eclipse environment) for Eclipse for visually edit and preview Android layouts

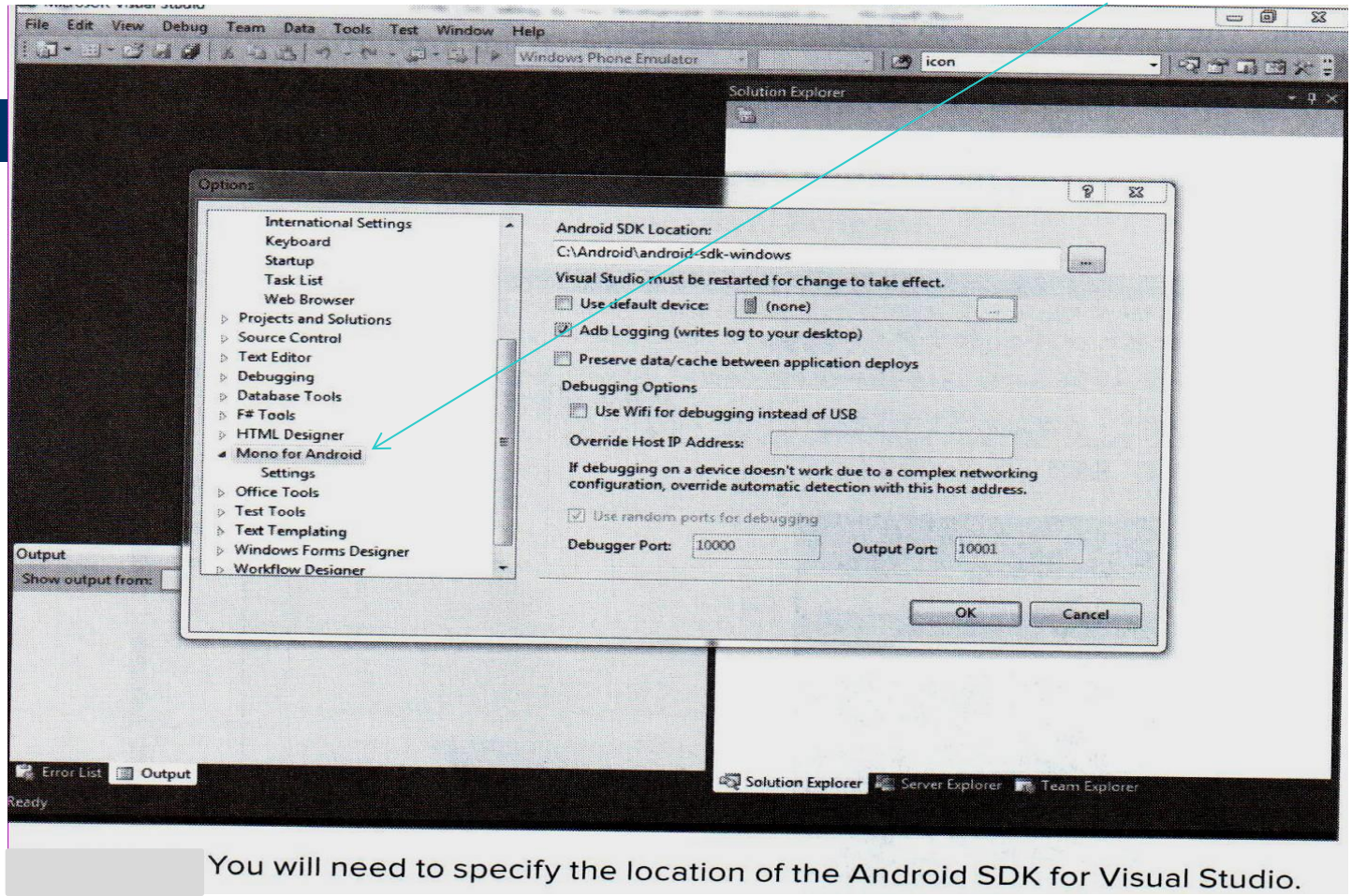
4.1 install Eclipse Classic (free)
4.2 start Eclipse →install new software
4.3. Installing Android plug-in:
click into developer tools



After you install the Eclipse Android plug-in, a number of developer

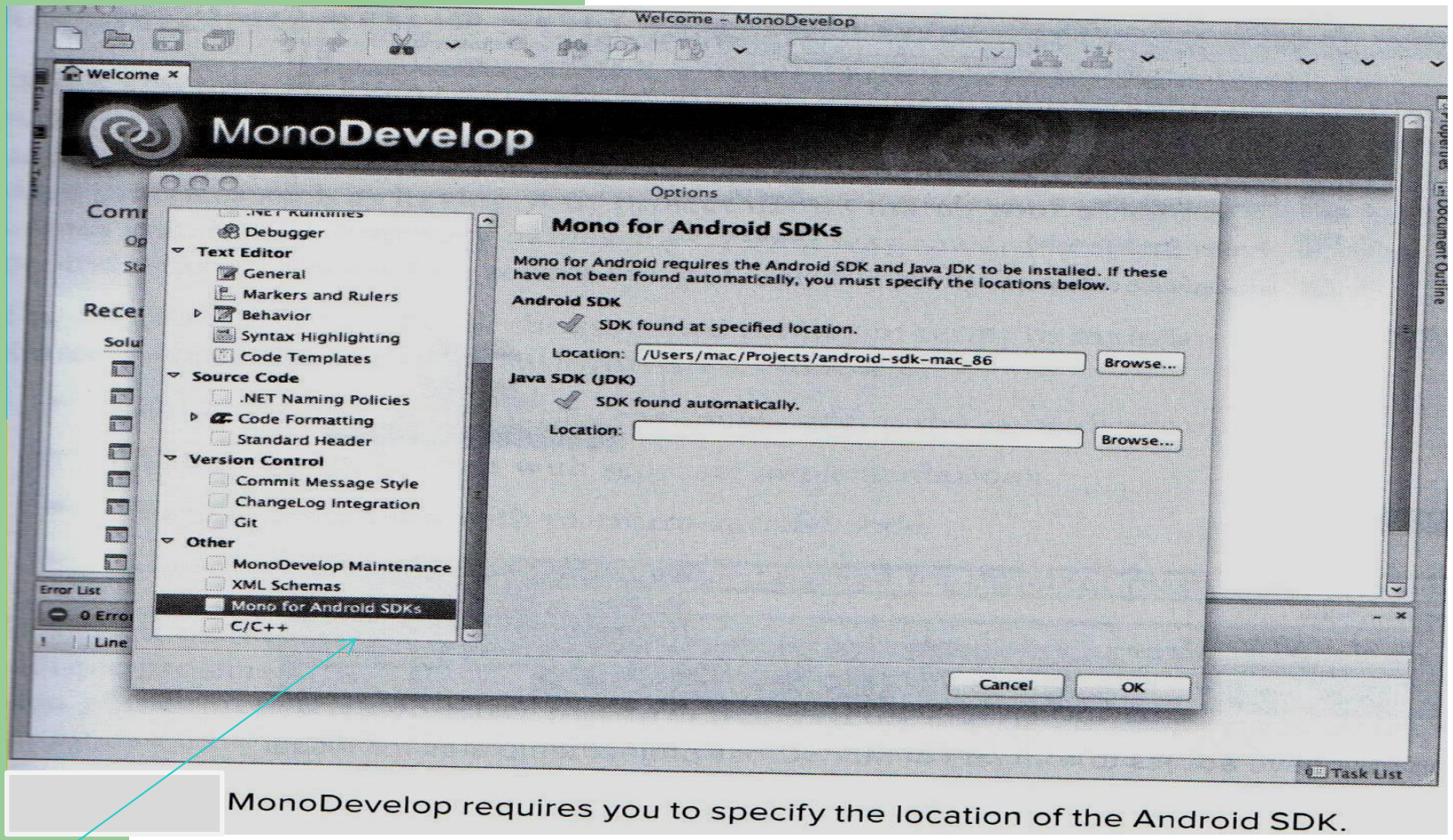
4.4 Installing Mono for Android

4.4a Configuring Mono for Android in Visual Studio (Tools → Options → Mono for Android)



You will need to specify the location of the Android SDK for Visual Studio.

4.4b Configuring Mono for Android in MonoDevelop (the same as in Visual Studio)



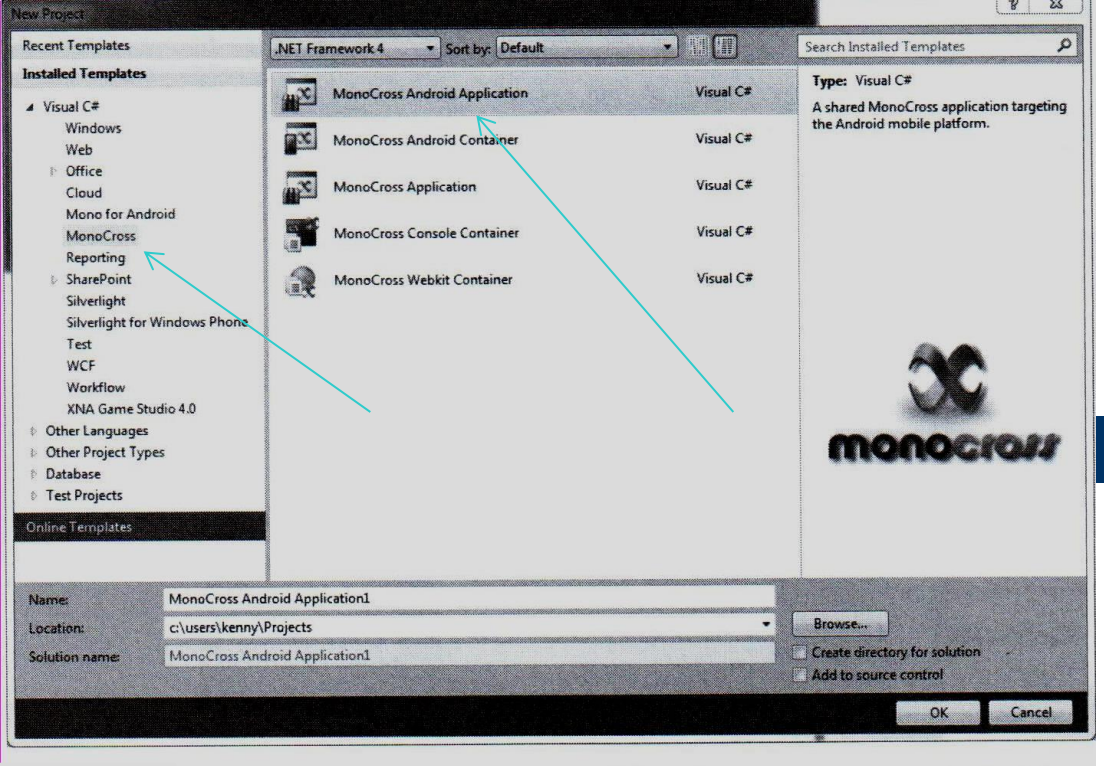
MonoDevelop requires you to specify the location of the Android SDK.

4.5a. Installing MonoCross Project Templates for Visual Studio

When using Mode View Controller (MVC) soft. Architecture & cross-platform development, you need a framework as MonoCross . You can use a number of templates and utilities.

MonoCross is installed on Visual Studio.
Templates are available for Android, Windows Phone, WebKit and Console MonoCross bindings.
There are many class templates , exposed for these platforms also.

So, download MonoCross Templates, unzip and install them;
start Visual Studio and using Tools → Extension Manager incorporate templates:



You now have access to a variety of MonoCross project templates for Visual

4.5b Installing the MonoCross Templates for MonoDevelop

Templates for Mono for Android and MonoTouch MonoCross bindings are available

Some theory needed to develop cross- platform apps

(about MonoCross, Model-View-Controller (MVC), Separate Interface Patterns)

- How to solve problems in app for different platforms?
- Which framework to chose: Java and Android, Mac and iOS, .NET and Windows Phone?
- The principle of : One language, one platform?!

- Is **HTML 5** the answer of cross-platform mobile problems? Exists HTML5 frameworks as

  Sencha
  Touch or
  JQuery Mobile

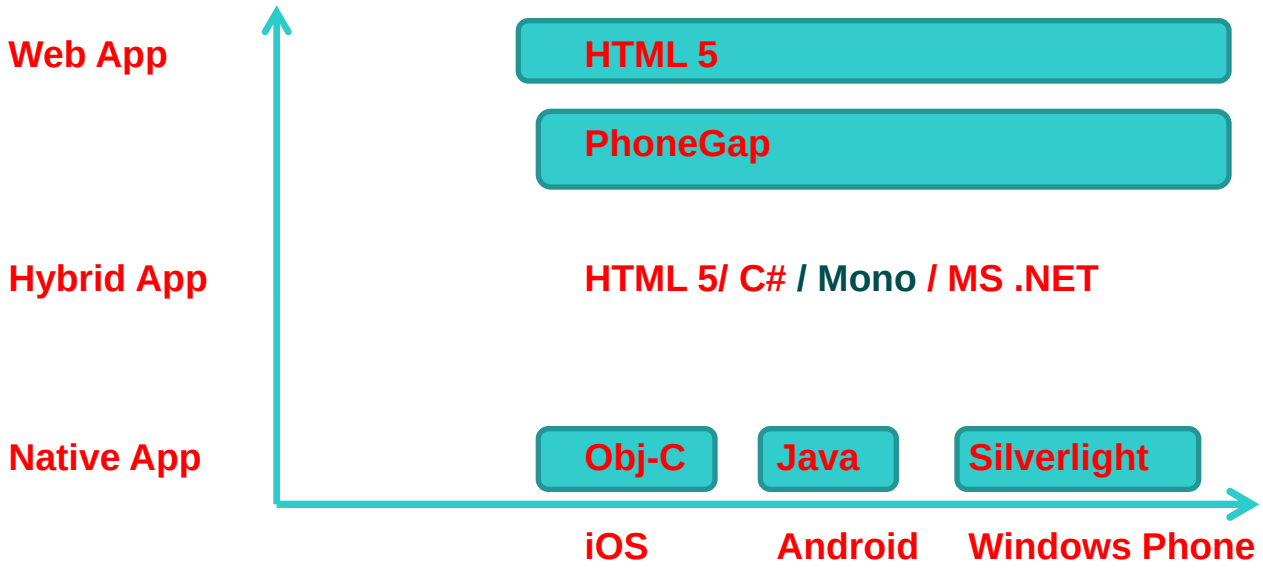
that gives impressive results.

But HTML 5 can not resolve all the problems: work with native phone features – GPS, camera, accelerometer…, it works in a disconnected manner but is difficult to support offline transactions without some code outside the browser…

- HTML 5 is not able directly to use the **native APIs**. Must be written a second layer – usually in JavaScript – that expose the native functions to the browser.

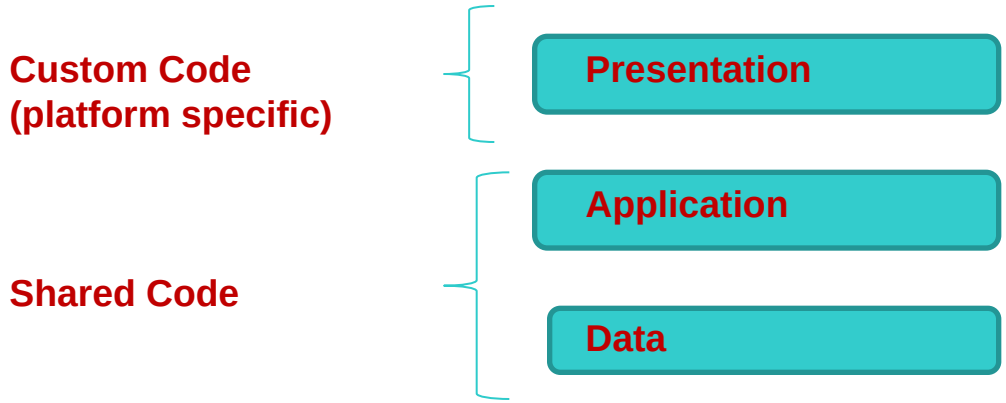
Code portability with Mono

For programmers with C# and .NET, we have now MonoTouch and Mono for Android, as well web app using HTML5 and ASP.NET.
Programmers have a new choice – Hybrid Apps

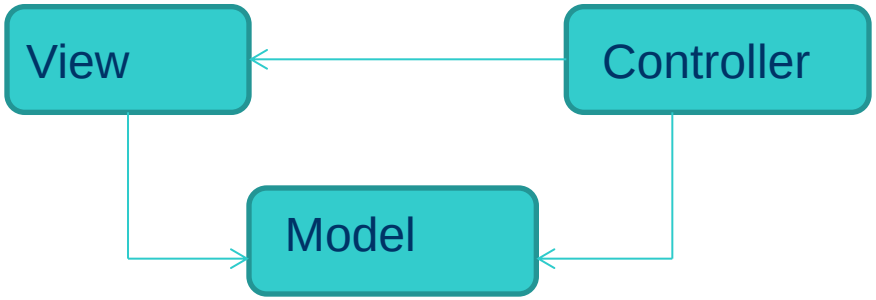


Developing for multiple platforms

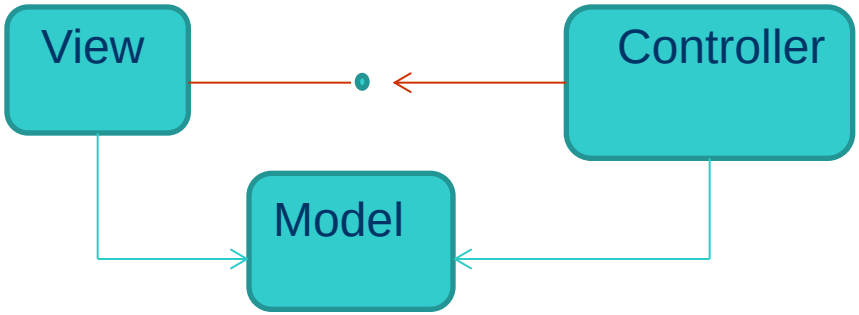
Separating the User Interface



Model-View-Controller (MVC) and Separated Interface patterns
(separation of the business and data logic from presentation layer)



Traditional MVC pattern



MonoCross MVC pattern includes a separated Interfaces (Separated Interface definition)

Defining a simple model

```
public class Customer
{
  public Customer()
  {
    ID = "0";
    Name = string.Empty;
    Website = string.Empty;

    PrimaryAddress = new Address();
    Addresses = new List<Address>();
    Contacts = new List<Contact>();
    Orders = new List<Order>();
  }

  public string ID { get; set; }
  public string Name { get; set; }
  public string Website { get; set; }
  public string PrimaryPhone { get; set; }
  public Address PrimaryAddress { get; set; }
  public List<Address> Addresses { get; set; }
  public List<Contact> Contacts { get; set; }
  public List<Order> Orders { get; set; }
}
```

An example program within MVC model:

- You want to work with a customer in a management system

Here is the **Model** object (business objects being acted in the app)

(must be in : project_name/Model/Customer.cs file)

Now you need to create the **controller** that will load the model and make some decisions about processing of the model for presentation in your views:

```
namespace MonoCross.Navigation
{
    public interface IMXController
    {
        Dictionary<string, string> Parameters { get; set; }
        String Uri { get; set; }
        IMXView View { get; set; }
        Type ModelType { get; }
        object GetModel();

        string Load(Dictionary<string, string> parameters);
        void RenderView();
    }

    public abstract class MXController<T> : IMXController
    {
        public string Uri { get; set; }
        public Dictionary<string, string> Parameters { get; set; }
        public T Model { get; set; }
        public Type ModelType { get { return typeof(T); } }

        public virtual IMXView View { get; set; }

        public object GetModel() { return Model; }
        public abstract string Load(Dictionary<string, string> parameters);
        public virtual void RenderView() { if (View != null) View.Render(); }
    }
}
```

The controller + interface definition for him:
the controller definition contains methods used by MonoCross framework to load controllers, process model logic and render views:

This is a general Controller Class

-business and data logic for the Model type.
- <T> is now <Customer>

Must be in: …/ MXController.cs

For your Company class you have to define own controller: Into ../Controllers/CustomerController.cs

Now you need a **class CustomerController : MXController<Customer> ,**
based on the abstract class above- for your Company system.

There in the **overridden Load()** method must be the logic for initialization and first presentation of the Company object.

The Load() method receive a ‘ dictionary ’ of parameters with model-specific information like “Custome ID” and after the load, retrieving the data , sets the ‘model‘ property of the ‘CustomerController‘ to a new “Company“ object instance.

There is a set of methods like **UpdateCustomer(), DeleteCustomer()..** also defined into these class .

String ‘’perspective” is ‘’switched” and searched for for different cases in this function also. Cases correspond to all ‘’action” like : GET, EDIT, DELETE, CREATE, UPDATE for the model.

Bulding a Platform-Specific View

(to display customer's information for current OS)

MonoCross MVC pattern separates the presentation layer to the application code by an IMXView interface and his abstract implementation MXView<T> class.

Use the MXView<T> - or MXConsoleView<T> to create a view for the Customer model class:

```
class CustomerView : MXConsoleView<Customer>
{….
public override void Render();
…
}
```

The Render() is the place you define the logic to display your model customer instance to the user.

into ../Views/CustomerView.cs

So, till this moment you must have:

- A **company model** class;
- A **controller** to load a company from the data-store;
- A **view** to display the company information in the presentation layer.

URI (Uniform Resource Identifier)-based navigation

(we want to produce cohesive app , that's able to deploy to multiple device platforms. URI structure associates each controller with URI-endpoint, that uniquely identify its place in the workflow)

- Building the shared (multi - OS) Application

First: there is an abstract MXApplication class to be inherited for all MonoCross shared apps.

pubic class App : MXApplication
{… **public override void OnAppLoad() {**
..

Into: …/App.cs

//is the place to display different customers, for which you have created a controllers. You need to Add
// the controller to NavigationMap for each. At runtime the navigation framework extracts CustomerId
// and places it in the controller arg passed for CustomerController.Load().
// You can pass an Action as well to the controller. This is when you want to use the same
//business logic from the controller and specify new action like 'Create', 'Delete' etc.
//Actions are performed in corresponding 'çases' in CustomerController.Load() implementation.

NavigationMap.Add("Customers", new CustomerListController());

CustomerController customercontroller = new CustomerController());
NavigationMap.Add(""Customers/{CustomerId}, CustomerContoller);
NavigationMap.Add(""Customers/{CustomerId} / {Action}, CustomerContoller);
… }

Adding a platform Container

(inside you define views for every platform, so shared controllers render them as defined into the app workflow)

First you have to inherit from abstract class **MXContainer for each platform supported.**

You put there your application views:

```
public class MXConsoleContainer : MXContainer
{ public MXConsoleContainer( MXApplication theApp) : base( theApp){}
```

// there is the place for initialization of the app; instantiating the controller (so views are registered)}

Having a container class, in the Main() you have to initialize it (or them) and to add views:

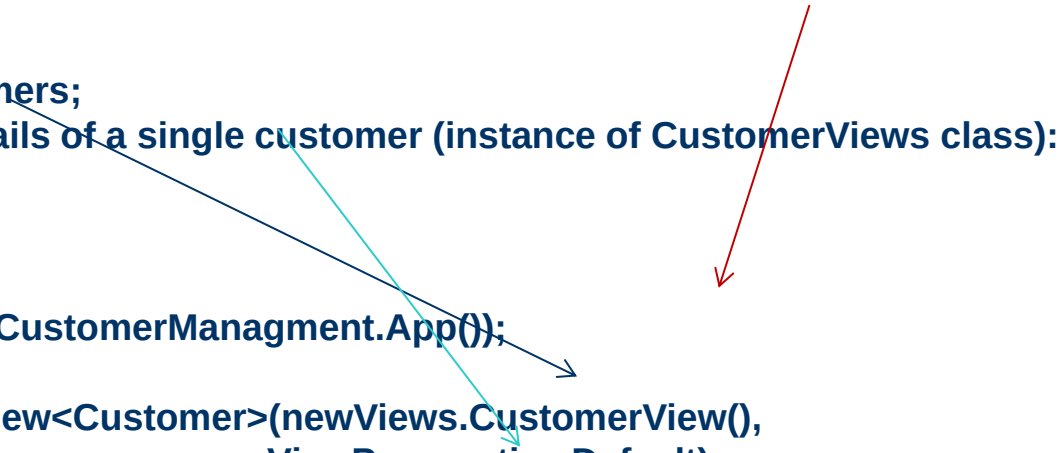
1)Create an instance of the shared app;

2)register the views by AddView(), passing instance of each view +ViewPerspective name:

For the example, you have 2 views:

first -displays a list of customers;

the second - displays the details of a single customer (instance of CustomerViews class):



```
class Program
{ static void Main(string[] args)
        {  MXContainer.Initialize(new CustomerManagment.App());

          MXConsoleContainer.AddView<Customer>(newViews.CustomerView(),
                                                ViewPerspective.Default);
         MXConsoleContainer.AddView<Customer>(newViews.CustomerEdit(),
                                                ViewPerspective.Update);
……      }
```

- Each view corresponds to a particular model type: for example one displaying list of orders, another - displaying order's details;
- To register views for the app, you have to specify the purpose for each view – in MonoCross that's done with '**perspectives**" ;

- The **view perspective** is simply a string describing the intended purpose of your view:

```
public static class ViewPerspective
{       public const string Default = "";
        public static string Delete = "DELETE";
        …
}
```

In MXViewPerspective.cs

- The ViewPerspective class defines string constants for all views.

- In the CustomerController.Load() (mentioned before) you defined the logic for them (in corresponding cases):

```
public override string Load(….)
{
// get the current action
switch(action)
{… case "EDIT":
        Model = new Customer();
        perspective = ViewPerspective.Update;
        // open form EditCustomerView to input new customer info , for example
….// other cases following …  }
```

The link between the controller and the view is

the model

and

the view perspective

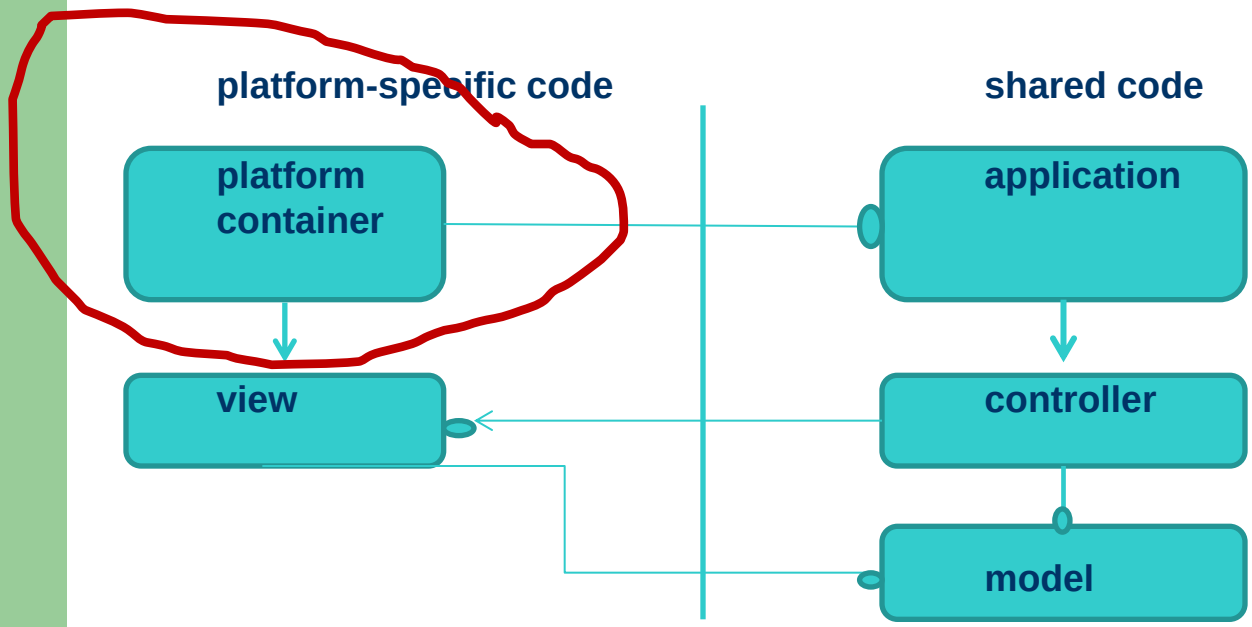
One option for data loading is to kick off data load in the controller and let the view pick it up when the Render() method is called.

In the case of Customer List Controller, there would be no logic executed in the controller.

The default view perspective could be used and execution passed on.

More about building a MonoCross Container

- How to tie together models, controllers and views?
- How to define platform-specific views?
- How to implement navigation between views?



1. a simple example -(console or text/view based) application

Must start with main():

initialize (instantiate an app object), then proceeds your first view)

```
namespace CustomerManagement.Console
{    class Program
     {

          static void Main(string[] args)
          {
               // initialize container
               MXConsoleContainer.Initialize(new CustomerManagement.App());
               // customer list view
               MXConsoleContainer.AddView<List<Customer>>(new
                     Views.CustomerListView(), ViewPerspective.Default);
               // customer view and customer edit/new
               MXConsoleContainer.AddView<Customer>(new
                           Views.CustomerView(), ViewPerspective.Default);
               MXConsoleContainer.AddView<Customer>(new
                            Views.CustomerEdit(), ViewPerspective.Update);
               // navigate to first view from the specified list
     MXConsoleContainer.Navigate(MXContainer.Instance.App.NavigateOnLoad);
          }
     }
```

Building the customer ListView

```
public class CustomerListView : MXConsoleView<List<Customer>>
  {
    public override void Render()
    {
       // Output Customer List to Console
```

All views implement IMXView which allows the container to track views, initialize view with its model and tell the view to render itself.

All platform containers provide basic template class for the implementation of views for this platform.

```
……

  this.Navigate(string.Format("Customers/{0}", Model[index - 1].ID));
…
}}
```

Building the customer View - **procedure,** the same as previous

Building the customer EditView - **procedure,** the same as previous

2. Implementing an iOS platform container

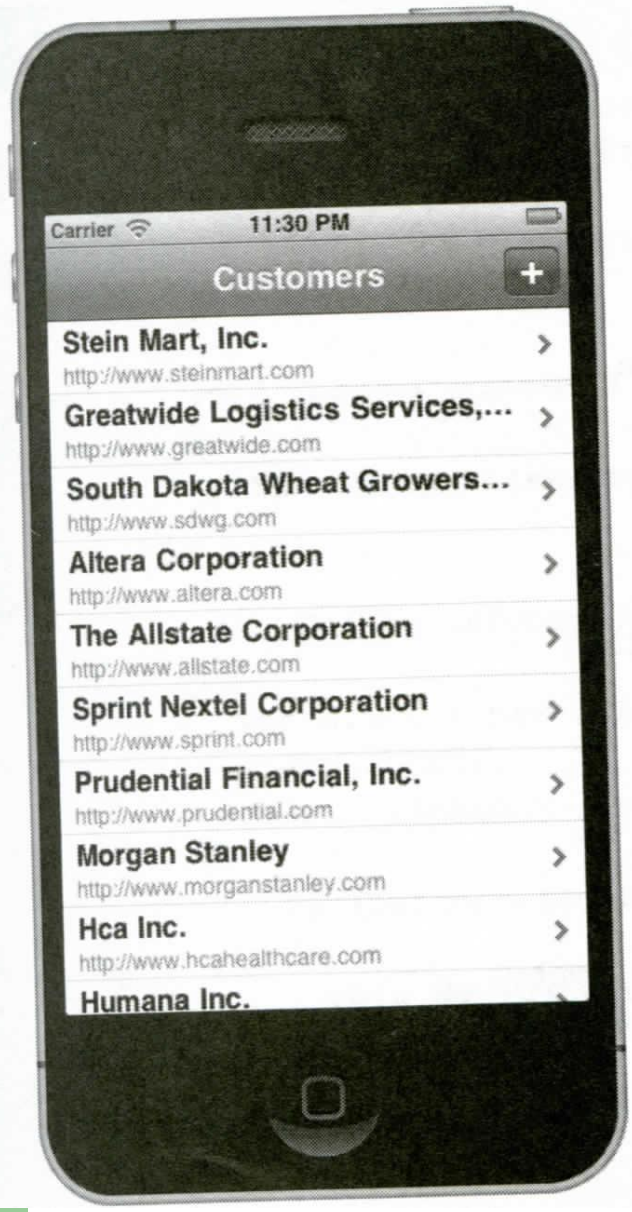
MonoTouch for iOS is like a slimmed-down version of .NET desktop + libraries of native iOS APIs that Apple provides to C/Objective-C developpers (Cocoa Touch).

That's not an abstract cross-platform UI. Instead this provides binding capabilities to the native API of the device.

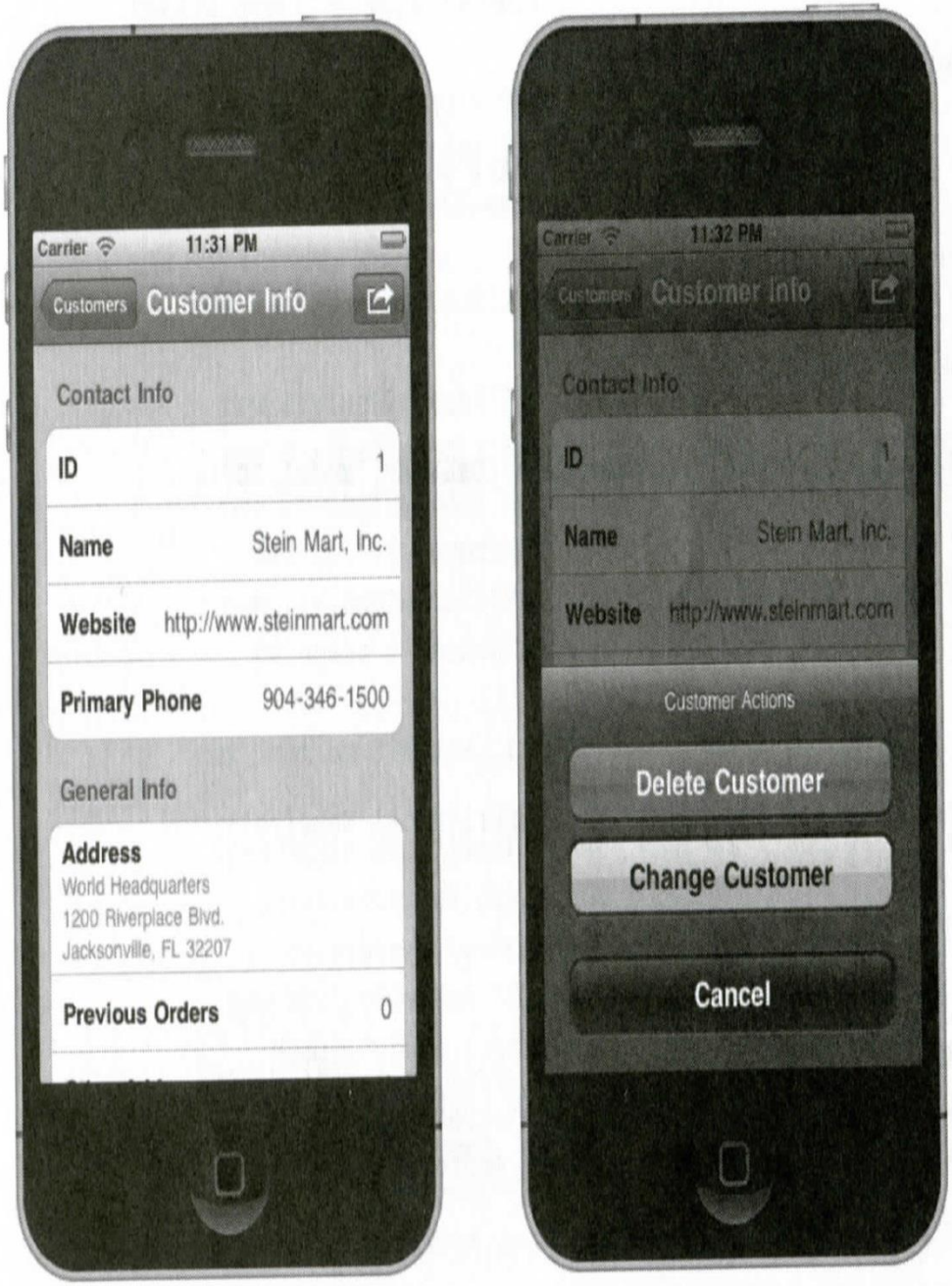
There is a binding from MonoTouch to Cocoa Touch classes (only a wrapper).

- First of all you have to write a code to **initialize the container** (MXTouchContainer class) and AddView() for it.
- After you have to write code for different **application's views** (for example: ListView, Customer View, Customer Edit View);

there are MXTouchView… base classes for the purpose.



iOS Customer List View



iOS Customer View

3. Implementing an Android container (with Mono for Android)

1 . Initialize the Container. Android Application consist of 1 or more activities. Any of them can be the entry point. The developer must specify which activity is the entry point, there are no main.

By property you specify the main activity class and describe functionality for events (On..() methods).

Using MXDroidContainer.AddView<View_Name>(..) you add views.
Using MXDroidContainer.Navigate(..) you change views.

2. You have to **describe All Views** successively naming by property like this:

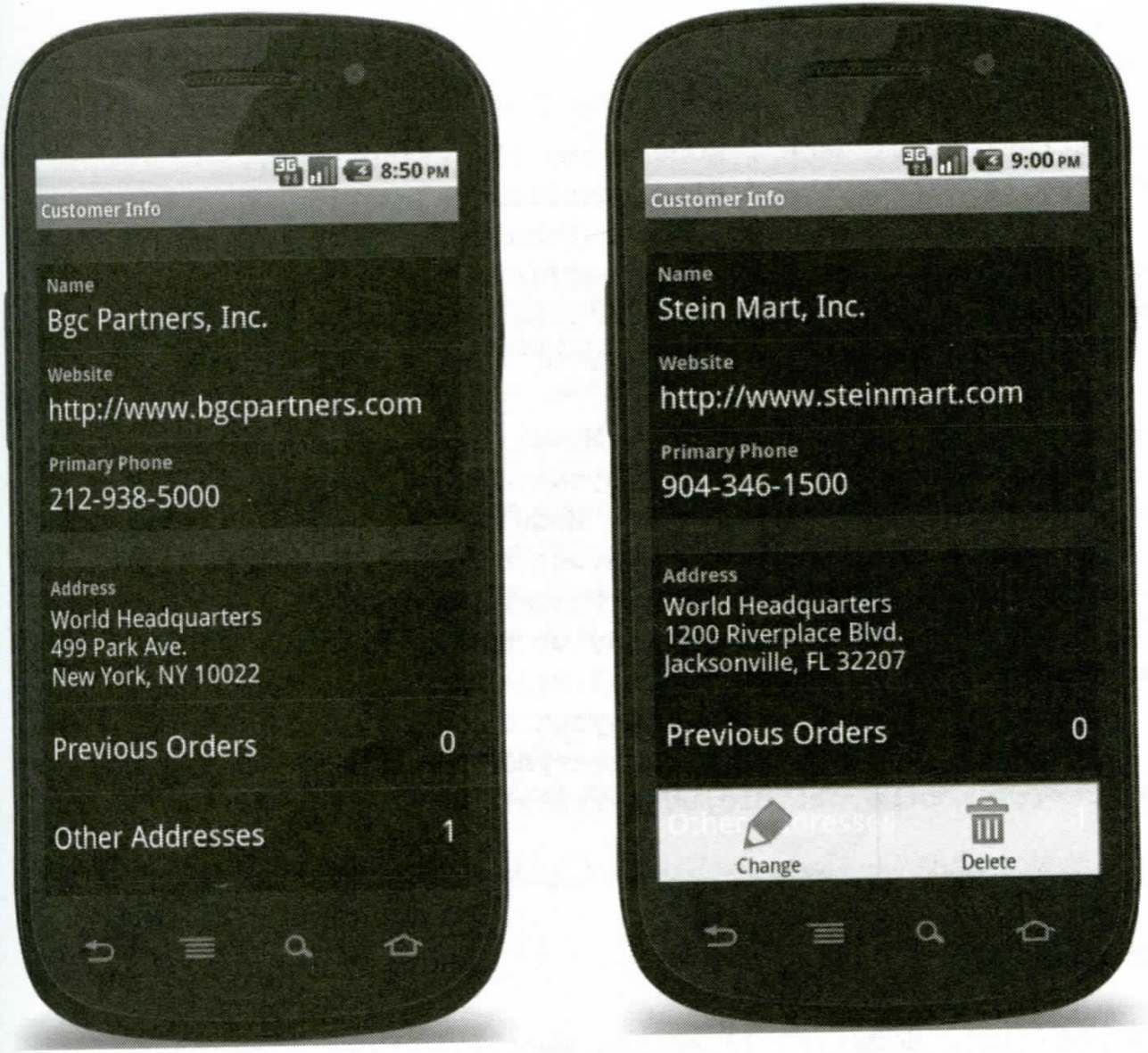
```
[Activity(Label = "Customert List" , Icon = "@darwable/icon")]
```

having activity name for the view, you are ready to create class and methods for it.
For example :

- ListView,
- CustomerView,
- CustomerEditView

MonoCross provides base classes from which you can derive your views.

Android Customer List View



Android Customer View

4. Implementing a Windows Phone Container

Windows Phone is version of Silverlight. Every page needs its XAML file describing view layout, formatting, data binding . Dynamic views are not supported.

Execution flow is message driven. Starting point is in Application derived class (App).

1. Initializing.

You have to write a class App : Application {..}
Like in the previous into this class must be :

```
MXPhoneContainer.Initialize(..);
MXPhoneContainer.AddView<View_Name>(..); …
MXPhoneContainer.Navigate(..);
```

Class Methods must be described.

2. Binding Views:

Description is in XAML files + into a normal file with the class methods code for every view.

Windows Phone
Customer List View

Windows Phone
Customer View

5. Implementing a WebKit container

MonoCross include capability to create container implemented for MS ASP.NET MVC framework.
The WebKit container runs on the server and not on the client (unlike other platforms like jQuery and jQTouch).
Views and controller render an HTML for every page sent down to the client

1. **Initializing** a Container with WebKit (occures during the Session Start and runs for each new client that connects to the web server)
   Like ASP.NET Global.asax.cs file is needed for each new Session
   code like this exists:

   ```
   MXWebKitContainer.Initialize(…);
   MXWebKitContainer.AddView<View_Name>(..);
   ```

2. **Building Views** with WebKit
   .html file (Root.html) exists for each interface. There are HTML, CSS, JavaScripts.

   MonoCross WebKit container provides 2 base classes from witch you derive you views:

   MXWebKitView<View_name> and MXWebKitDialogView<View_Name>.

In this classes (Methods exist for this) you define the content of the correspondent HTML. Example:

```
button.Controls.Add(image);
a.Controls.Add(em);
```

C# is used

WebKit
Customer List View

WebKit
Customer View

Develop (HTML5 +) Phone Apps with Apache Cordova

Apache Cordova is a framework for creating cross-platform mobile applications using HTML5 and JavaScript

All these smartphones have a highly capable browser, in many ways more capable than their desktop counterparts. Modern smartphones allow you to create applications that run within the browser using a combination of HTML5, JavaScript and CSS. With these technologies you can potentially write a single browser-based application that runs across a diverse range of smartphone devices.

You can create an HTML5-based mobile application by creating a public Web page with JavaScript and HTML5 content and directing people to the hosting URL.

However, there are a couple of problems with this approach.

- the distribution model through online marketplaces and stores. You can't submit the URL that hosts your Web app to a marketplace, so how can you monetize it?
- The second problem is how to access the phone's hardware.

Apache Cordova (or just Cordova) is a free and open source framework that solves both of these problems.

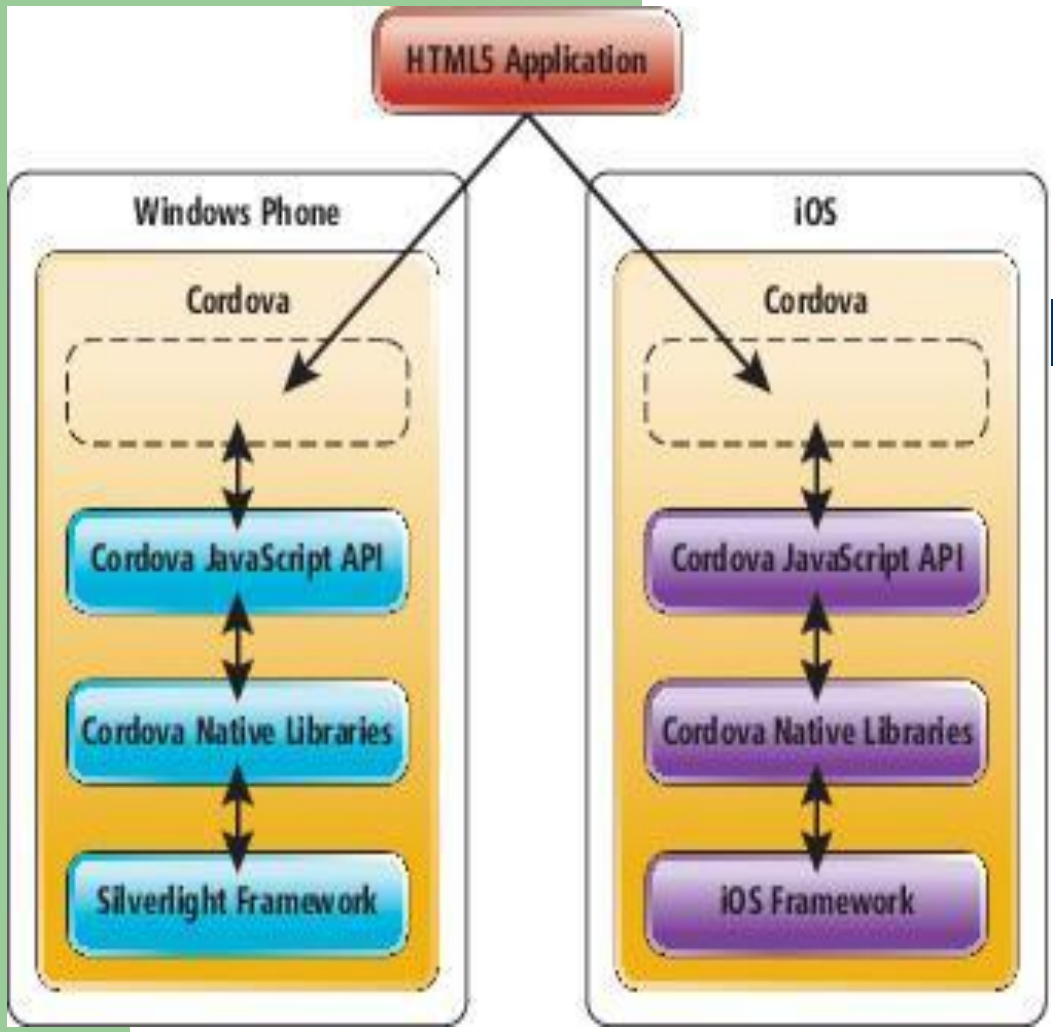
Cordova started life as PhoneGap, which was developed by Nitobi. In October 2011 Nitobi was acquired by Adobe Systems Inc., with the PhoneGap framework being open-sourced under the Apache Software Foundation and rebranded as Cordova.

Cordova provides an environment *for hosting your HTML5/JavaScript content within a thin native wrapper*. For each smartphone OS, it uses a native browser control to render your application content.

With Windows Phone, your HTML5 assets are packaged within the XAP file and loaded into isolated storage when your Cordova application starts up. At run time, a WebBrowser control renders your content and executes your JavaScript code.

Cordova also provides a set of standard APIs for accessing the functionality that's common across different smartphones. Some of these functionalities include:

Application lifecycle events
Storage (HTML5 local storage and databases)
Contacts
Camera
Geolocation
Accelerometer



Cordova is a cross-platform technology, you typically develop using your editor or IDE of choice, so an **iOS** developer would develop a Cordova application in Xcode and an **Android** developer would most likely use Eclipse.

Cordova also has a cloud-based build service called Build (build.phonegap.com), where you can submit your HTML5/JavaScript content. After a short time it returns distributions for most of the Cordova-supported platforms.

This means you don't need to have copies of the various platform-specific IDEs (or a Mac computer) in order to build your application for a range of platforms.

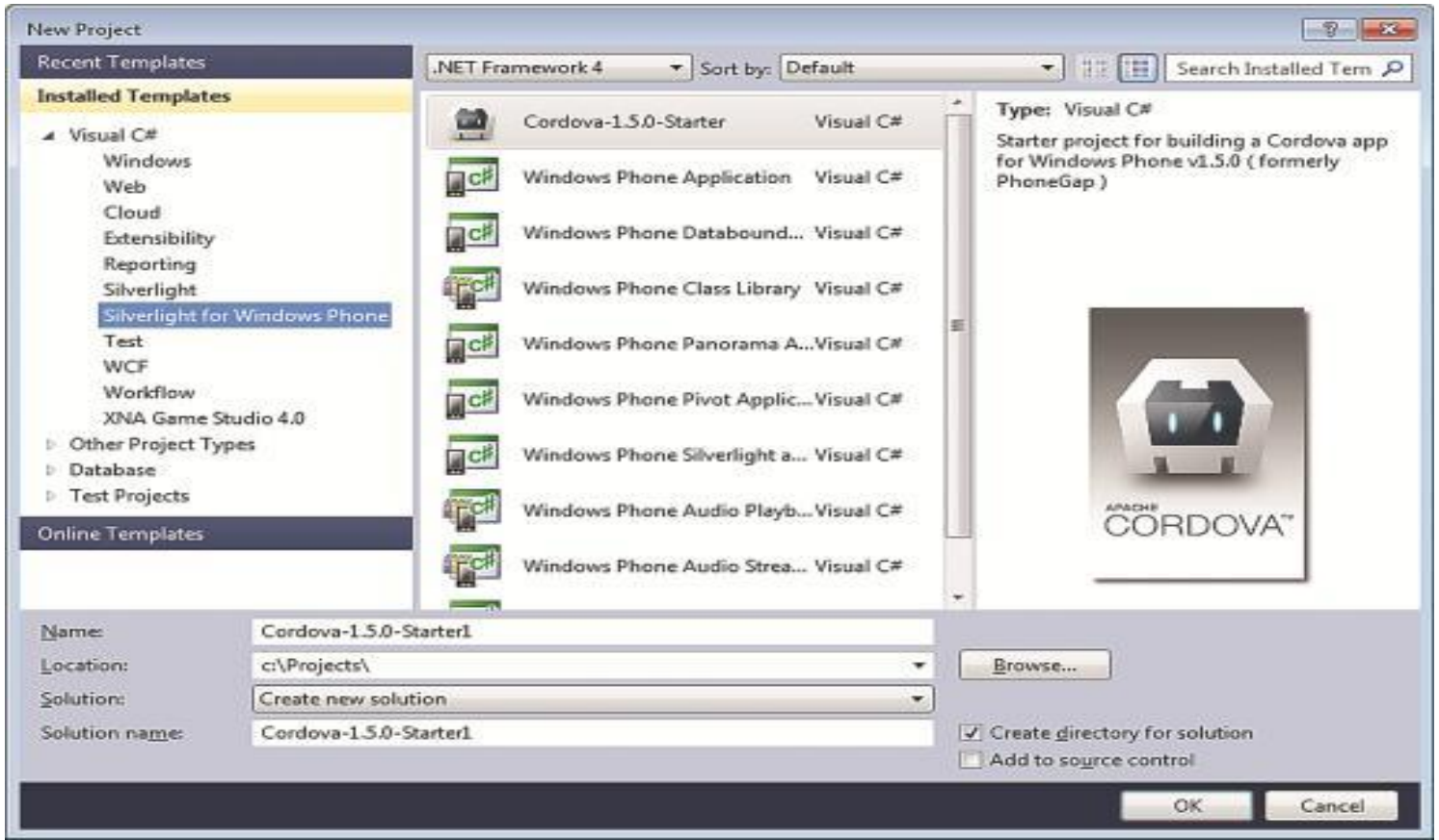
Acquiring the Tools for Cordova

you already have **Visual Studio**, the **Windows Phone SDK**.

You can obtain the latest **Cordova developer tools** from the PhoneGap Web site (phonegap.com), although future releases will be distributed via **Apache** . The download includes the templates, libraries and scripts required to develop Cordova applications.

Once you've downloaded the Cordova tools, follow the Windows Phone Gap Started Guide (phonegap.com/start#wp) and install the Visual Studio template.

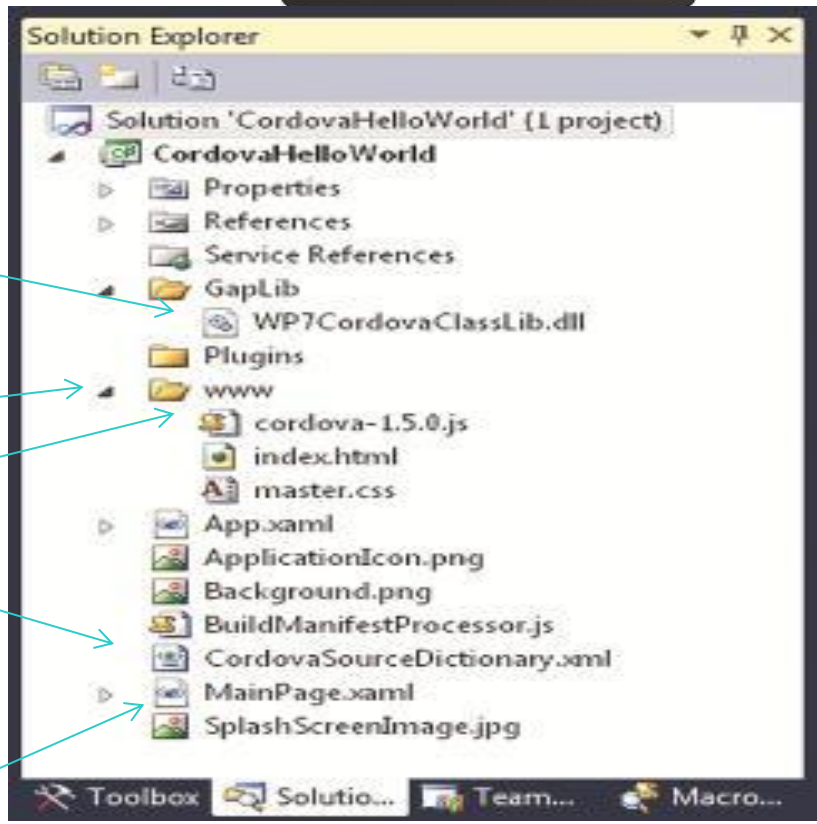
Creating a "Hello World"-style application is simple:



The Cordova Template Application Running on an Emulator:

The Anatomy of a Windows Phone Cordova Application

- **GapLib/WP7CordovaClassLib.dll** is the Cordova assembly. This contains the Windows Phone native implementation of the Cordova APIs.
- **www** is the folder where you place your application assets, HTML5, JavaScript, CSS and images.
- **www/cordova-1.5.0.js** provides the Windows Phone implementation of the Cordova JavaScript APIs.
- **CordovaSourceDictionary.xml** is a generated XML file that lists all of your application assets. When your application first launches, this XML file indicates the files to be loaded into isolated storage.



The **MainPage.xaml** file contains an instance of the CordovaView control, a user control that contains a WebBrowser control:

```
<Grid x:Name="LayoutRoot">
    <my:CordovaView Name="PGView" />
  </Grid>
```

When the app starts, the CordovaView control takes care of loading your application assets into local storage and navigating to the **www/index.html** file, thus launching your application.

Developing Cordova Applications

You can add your **HTML, JavaScript and CSS files to the www folder** and they'll be included in your project and accessible via the browser control when your application executes. You can use any of the standard JavaScript/HTML5 libraries or frameworks in your Cordova application, as long as they're compatible with the phone's browser.

The Cordova APIs are documented on the Cordova Web site. One important thing to note is that you must wait for **the device-ready event** before making use of any of the other API methods. If you inspect the index.html file generated from the template, you can see that it waits until the device is ready before updating the UI:

```
<script type="text/javascript">
document.addEventListener("deviceready",onDeviceReady,false);
function onDeviceReady()
        {document.getElementById("welcomeMsg").innerHTML
                += "Cordova is ready! version=" + window.device.cordova;
console.log("onDeviceReady. You should see this " +"message in Visual Studio's output window.");
        }
</script>
```

Single-Page or Multipage Application Architecture

When building Cordova applications, you can employ two distinct patterns:
Multipage applications: In multipage applications, multiple HTML pages are used to represent the various screens of your application. Navigation between pages uses the standard browser mechanics, with links defined by anchor tags. Each HTML page includes script references to the Cordova JavaScript code and your application JavaScript.
Single-page applications: In single-page applications, a single HTML file references Cordova and your application JavaScript. Navigation between the various pages of your application is achieved by dynamically updating the rendered HTML. From the perspective of the phone browser, the URL remains the same and there's no navigation between pages.

The multipage approach has some disadvantages: First, when the browser navigates from one page to the next, it has to reload and parse all the JavaScript associated with the new page. There's a noticeable pause as the Cordova lifecycle. Second, because your JavaScript code is being reloaded, all application state is lost.

The single-page pattern overcomes the issues associated with the multipage approach. The Cordova and application JavaScript code is loaded just once, resulting in a more responsive UI and removing the need to pass application state from one page to the next.

The Cordova framework makes it possible to create HTML5-based applications for **Windows Phone**. It's also possible to mimic the native look and feel using simple HTML5 and CSS techniques to **an iPhone** or **Android** phone without modification.

As a demonstration of the versatility of this approach, here are an iOS version of an application, using jQuery (to integrate with DOM) and a Phone version of the same application.