# Windows Phone 7 Guide for iPhone Application Developers


Microsoft
3/25/2011
Rev 3.0

Windows Phone

Windows Phone

# About this Development Guide


If you have been developing iPhone applications and are interested in building your applications for Windows Phone 7, this guide is for you.

The guide will cover what you need to know to add Windows Phone 7 development to your skill set, while leveraging what you have already learned building iPhone applications.



Windows Phone

# Chapter 1: Windows Phone 7 Platform introduced to iPhone application developers

## New Beginning

On October 11th Microsoft announced the release of Windows Phone 7 on 10 devices from a variety of manufacturers all over the world. Almost 2000 applications are already available on the Windows Phone 7 marketplace.

For Windows Phone 7, Microsoft went back to the drawing board to figure out what phone users really want, and built a phone from the ground up. The OS, the user experience and the application development platform have all been engineered with users in mind. The revenue opportunities in the Windows Phone marketplace, accompanied by a great set of development tools, make WP7 a very attractive destination for developers to build applications and games.

## Developer Tools

In early September, Microsoft released a set of tools for Windows Phone 7. This toolset is free and can be downloaded from here. The toolset includes:

- An IDE (for developers) : Visual Studio Express for Windows Phone,
- A User Interface design tool (for designers): Express Blend for Windows Phone,
- Frameworks: Silverlight for Windows Phone and XNA Game Studio for Windows Phone
- And a Windows Phone 7 emulator to test and debug applications.

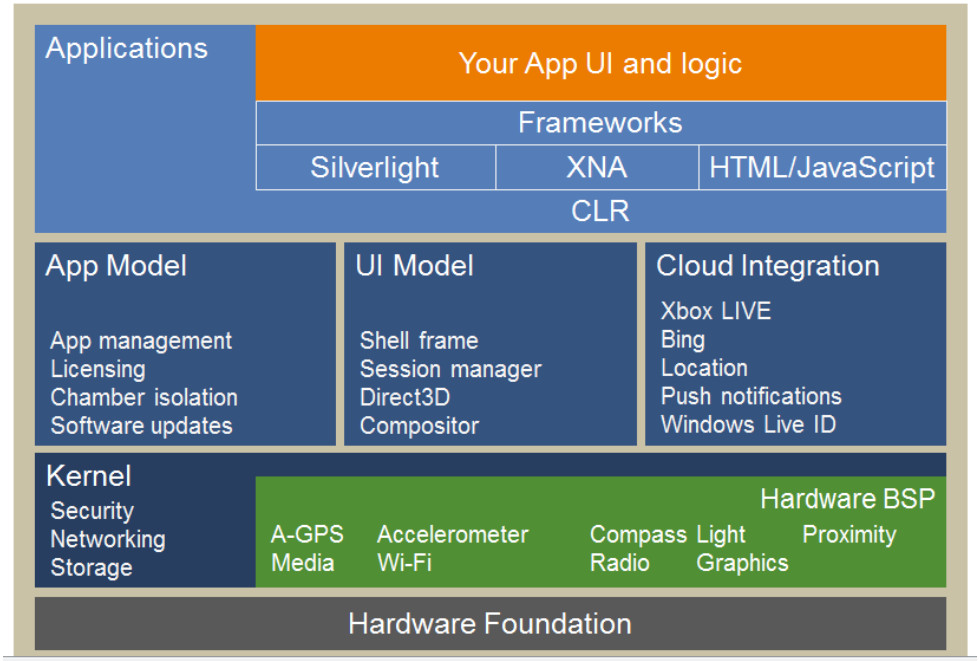
The tools are designed to let you develop consumer applications, business applications or games.

## Windows Phone 7 Architecture

Windows Phone 7 utilizes a layered architecture as shown below. In contrast to the iPhone OS, WP7 will run on multiple phones. To provide a consistent user experience and features that developers can rely on, it defines a minimum set of hardware specifications that all phones must meet.

They include an ARM7 CPU, a DirectX capable GPU, a camera, and a multi-touch capacitive display. Standard sensors include: an A-GPS, an accelerometer, a compass, proximity and light sensors. There are three standard physical buttons on the phone – back, start and search. As we will see in a subsequent chapter, these buttons provide an easy and natural navigation model for the user.

In WP7, Microsoft provides most of the device driver code. The device manufacturer has to write very little code specific to their device. This is expected to improve the consistency and quality across various devices. WP7 takes advantage of hardware acceleration through encapsulation layers such as DirectX or XNA.




WP7 applications use managed programming and run within sandboxed environments. Watch this MIX '10 presentation by Istvan Cseri , a Windows Phone 7 architect to get more details on the WP7 architecture.



Windows Phone

## Comparing the WP7 Programming Stack with the iPhone Stack

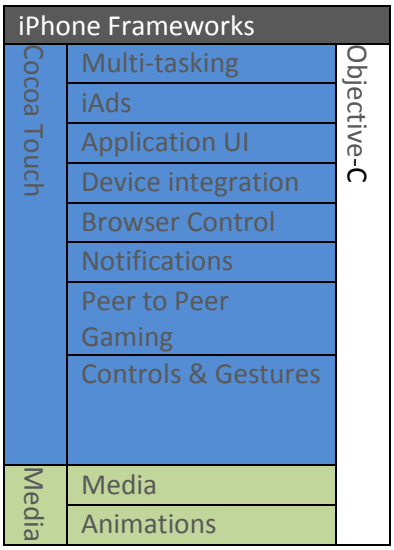
The App Model shown above provides services for managing the application lifecycle such as installation, and update. The UI model helps manage application user interface. Applications are built using various WP7 frameworks.

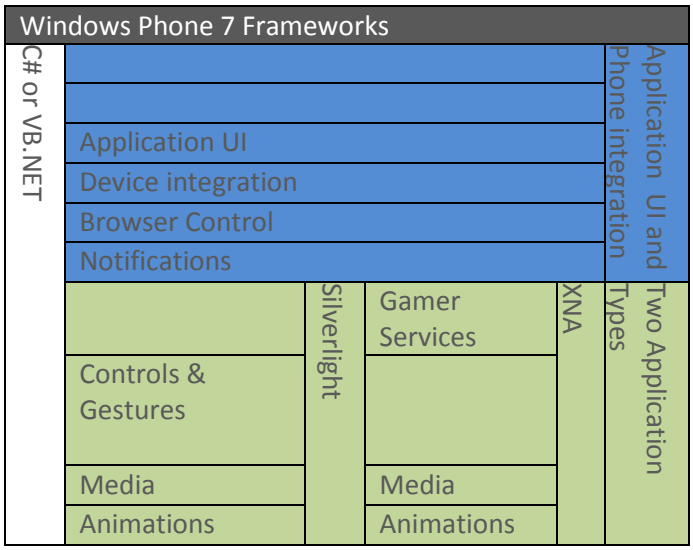
The following table gives an overview of the Windows Phone 7 frameworks that provide features comparable to the iPhone programming layers.

| iPhone Frameworks | Functionality | Windows Phone 7 Frameworks: |
|---|---|---|
| Cocoa Touch | Application UI, Device integration (sensors, camera) | WP7 Phone Framework, Silverlight controls |
| Media Layer | Graphics, Animation, Media | XNA for games or Silverlight media and graphics for others |
| Core Services layer | Base services, Networking, Text, XML, storage | Common Base Library |
| Core OS layer + iOS | | Window Phone 7 OS |

### iOS and WP7 Stacks side by side

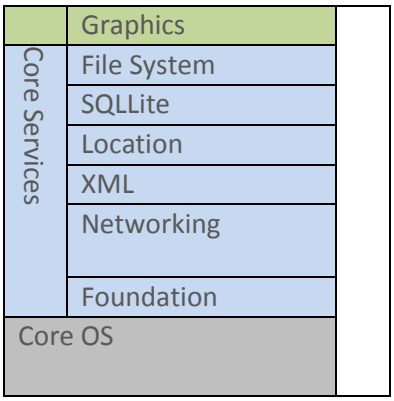
The following table provides a detailed look into the framework layers shown above. The left hand side shows the iPhone stack with the corresponding framework from Windows Phone 7 on the right. These frameworks can be grouped in three large buckets, namely, Application UI and Phone integration, Base services, with the OS layer underneath.

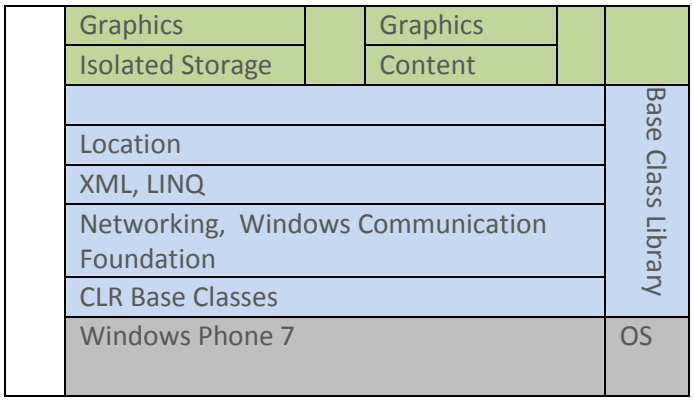





Windows Phone




## Managed Code only

On the iPhone, you have been using Objective-C. WP7 only supports "managed code" applications using C# or VB.net; there is no native access available to the system or the phone hardware. Execution of such code is managed by the .NET Common Language Runtime (CLR). One of the benefits is that CLR provides garbage collection - there is no memory management to worry about or pointers to take care of. The WP7 application stack is built on the .NET compact framework 3.7. The .NET compact framework is optimized for resource constrained devices and is designed to be portable across various hardware platforms.

## Base Services

WP7 Base Class Library classes roughly correspond to those provided in the Foundation framework in the iOS Core Services Layer. They include base classes, collections, threading, text processing and IO. The WP7 Base Class Library layer also includes networking stacks, such as HTTP and the Windows Communication Foundation (WCF). WCF provides an easy interface with XML and SOAP services across the web, with features supporting XML data transfer, serialization/deserialization and XML parsing. While WP7 does not have a local database such as SQLLite, developers can write SQL-like queries in C# using Language Integrated Query (LINQ) to query XML data, stored in isolated storage (see below), or in remote databases such as SQL Azure.

# Application UI and Device Integration

## WP7 – Two Options for Applications UI

If you are using the iOS Media layer frameworks, you have two stacks to choose from in WP7, namely, Silverlight and XNA. While you can use either, generally, it is recommended that you use Silverlight for consumer or business applications and XNA for games, although you can certainly also develop great games using Silverlight animation.

| | | |
|---|---|---|
| iPhone Applications: UI using views with navigation between them |  | WP7 Applications<br><br>Silverlight apps with pages connected by flows |
| iPhone Games:<br><br>2D or 3D games built with Quartz or OpenGL ES |  | XNA games with 2D / 3D graphics and Xbox connectivity |

## XNA for Games

XNA framework, originally developed for XBOX, provides hardware accelerated 2D and 3D rendering and bitmap graphics. XNA also provides gamer services such as authentication and connectivity with XBOX Live, as well as profiles and leaderboards. For a high performance game, XNA is the right option.

## Silverlight Controls and Media

If you have been using Cocoa Touch for controls and multi-touch, you will find a large set of Silverlight UI controls specifically designed for the phone and supporting multi-touch. Silverlight uses a declarative language called Extensible Application Markup Language (XAML) to specify user interfaces. Developers can use separate code-behind files, written in C# or VB.NET, to respond to events or manipulate the controls.

Silverlight provides high performance audio and video with variety of CODECs. It supports both vector and bitmap graphics with hardware acceleration.  As opposed to a file system, Silverlight provides sandboxed storage, called isolated Storage, to store the application-specific data. With the isolation of storage, one application cannot affect other applications that are running on the phone.

### Windows Phone Frameworks

If you need to use HTML in your application, you can use the IE-based browser control in your application for HTML UI. Windows Phone framework layer also provides interfaces to various sensors, such as the accelerometer or the camera. Similar to Apple's notification service, Microsoft provides a push notification service, called Microsoft Push Notification Service. In iOS 4.0, Apple introduced multi-tasking and iAds for advertisement support in the application. While multitasking is not available on Windows Phone 7, Microsoft has recently released Microsoft Advertising SDK for Windows Phone 7.

## Summary

In this chapter we looked the Windows Phone 7 architecture and the two programming stacks. Now that you have a high-level idea of how the WP7 programming stack maps to the iPhone stack, we are now going to go one level deeper. In the next chapter, we will look at the user interface guidelines of WP7 applications.

## Related Resources

To go deeper into the topic discussed, check:

1. App Hub – Central place for Windows Phone 7 development. Getting started, download tools and read all about Windows Phone 7 development
2. MIX '10 presentation on Windows Phone 7 Architecture by Istvan Cseri

Other Resources you may find useful:

1. Overview of the Windows Phone 7 Application Platform
2. Windows Phone 7 team blog.
3. Windows Phone 7 Programming: Programming guide and reference documents.

# Chapter 2: User Interface Guidelines

## A new UI paradigm

Microsoft's Windows Phone 7 uses a novel user interface called Metro. It sets itself apart with its clean and simple design and emphasis on color and typography.


In contrast with the application-focused design of the iPhone, WP7 uses an information-centric design. Instead of an array of application icons, the start screen of a Windows Phone consists of dynamic tiles that display critical information at a glance to the user. The tiles themselves are dynamic, in that they continuously portray the up-to-date status of the application. For example, they can show you the next appointment on your calendar, or the number of new emails waiting for your attention. Users can personalize their phone by pinning the tiles that they care most about.

WP7 introduces a new paradigm called "hubs". Hubs bring related information together. There are six hubs, namely, People, Pictures, Music + Videos, Marketplace, Office, and Games. The People hub, in the instance shown below, aggregates your address book contacts and Facebook friends.




Windows Phone

## Designing the Application Interface:

While the design of the Windows Phone 7 user interface is different from that of the iPhone, the core design principles are very similar. Like the iPhone, WP7 developers have to keep in mind the compact screen, lower CPU and limited memory while designing the applications. Users use one application at a time, with just one screen visible.

### Similar Application Design Goals

Usability and UI design are not afterthoughts, but are the primary goals behind applications on both the iPhone and WP7. Applications need to be simple and focus on key scenarios that most users care about.

### Visual Elements and Direct Manipulation

Like the iPhone, visual elements and direct manipulation of objects by touch are the key characteristics of the WP7 application. WP7 provides a complete set of UI controls designed for the phone. It utilizes the same set of core multi-touch gestures as the iPhone with similar semantics – these include tap, double tap, pan, flick, touch and hold, and pinch and stretch.

### Implications of the similarities for the developers:

For the most part, your application planning process will be similar on both platforms. While designing your WP7 application, you will focus on the same information that is critical to the user. Your key design principles from the iPhone application will get carried over: metaphors, direct manipulation with multi-touch, the need for immediate feedback and aesthetic appeal, will still remain the same.

## Application User Interface Design

While there are similarities in the design principles of the applications on both platforms, pay close attention to the user interface of the application for the WP7. It is best to take advantage of the unique features and strengths of WP7 platform.

For the interface to provide a consistent experience across applications, applications on WP7 need to adopt the new Metro design guidelines.

### Controls and the Application Interface

The WP7 development tools and SDK include a rich collection of Silverlight controls designed specifically for usability and aesthetics. While you can create your own controls, it is best to use the standard controls where possible. These controls respond to theme changes and provide the consistent user interface.

The following table shows the mapping between WP7 Silverlight controls and corresponding iPhone controls.

| iPhone control: | WP7 control: | Notes: |
|---|---|---|
| Text field | Text box | |
| Label | Textblock | |
| Search bar | Textbox + button | |
| Rounded Rectangle Button | Button | |
| Segmented control | Radio Button | |
| Activity indicator | Progress indicator | |
| Slider | Slider | |
| Progress View | Progress bar | |
| - | Multi-scale image | Image with zoom capability |
| - | Panorama | Panorama to display related content that spans display |
| - | Pivot | To provide different views on the data |
| - | Grid | To arrange other controls in a tabular form |
| - | Ink presenter | Surface for inking |
| Page indicator | - | |
| UISwitch | ToggleSwitch control | Available on Codeplex* |
| Date and time pickers | Datepicker / Timepicker | Available on Codeplex * |
| Picker | - | Use Silverlight WP7 template |

* ToggleSwith and Datepicker/Timepicker control are part of the Silverlight for Windows Phone Toolkit available on Codeplex: http://silverlight.codeplex.com/releases/view/55034

As you can see above, WP7 offers controls that correspond to almost all of the iPhone controls. While the look and feel is different, they provide similar functionality.

## New Controls

Windows Phone 7 introduces a few novel controls that have no counterpart on the iPhone. A multi-scale image, with image data at various resolutions, is appropriate for allowing the user when zooming into a photo. Panorama control is a multi-screen page and allows a page to span horizontally beyond the width of the phone. The people hub, shown above, is a great example of this control. It allows a large amount of related information to be presented. Pivot control, another novel control shown below, is useful to manage views and display information that is logically divided in sections.


## Notifications

Both iPhone and WP7 have notification services, but notifications play a key role in WP7. The tile notifications are what make the tiles come alive. They are used to display non-critical information without disrupting what the user is doing. If you are using an application badge on the icon in an iPhone, you can use a tile notification as a replacement. However, tiles have the ability to provide far more information, such as photos (see above).

The notification service can also display toast notifications that provide time sensitive information such as an SMS. The toast notifications are shown for about 10 seconds, but the user may choose to ignore them. These are different from the iPhone alerts, which a user must respond to.

| iPhone | Functionality | Windows Phone 7 |
|---|---|---|
| Icon badges | Non-critical information that user may not respond to | Tile notifications |
| - | Time sensitive data that user may not respond to | Toast Notifications |
| Alerts | Modal alerts that user must respond to | Application notifications |

### Tool and Tab bar vs. Application bar

As opposed to separate tool bar and tab bar, WP7 only sports an application bar. The application bar can include up to 4 of the most common views or application tasks. You can also use application bar menus for additional context-sensitive tasks. If you are using action sheets in your iPhone application, application bar menus will provide you with similar functionality.

| | | |
|---|---|---|
| Status bar | Information about device | Status bar |
| Navigation bar | Navigation, Title, Buttons for views or actions | Back button for back navigation |
| | | Page title |
| | | View and actions on Application bar |
| Tab bar | Alternate views | Application bar |
| Tool bar | Actions in the current context | Application bar |
| Action sheets | Context sensitive menus | Application bar menus |

## Comparing WP7 and iPhone Navigation

WP7 application is a collection of multiple pages. Like on the iPhone, the user navigates through different pages using widgets such as buttons and links. However, the two platforms differ in their back navigation.

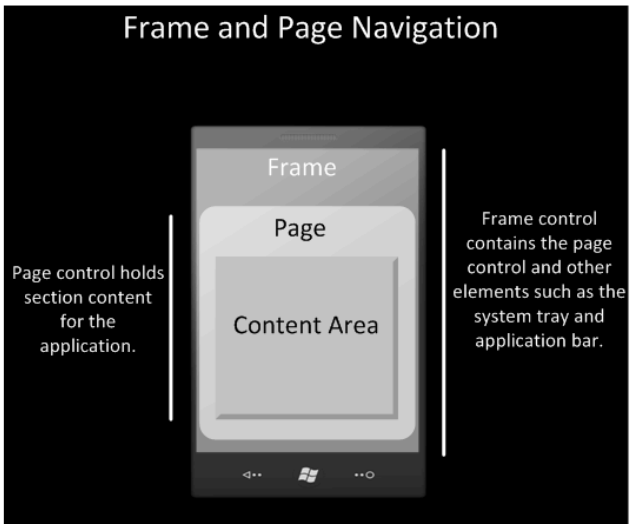
On the iPhone, developers need to implement the back functionality using the navigation controls on the navigation bar. On WP7, the hardware back button allows the user to navigate back between pages within an application, or across applications. It behaves much like the Back button in a browser. The Back button also closes menus and dialogs. As a developer, you should consider what the Back button means to your user and plan to override it appropriately. For example, you may decide to pause a game using the Back button.

The other two hardware buttons on the WP7 phone, namely, Search and Home, have fixed behavior.

## WP7 Frame and Page Structure

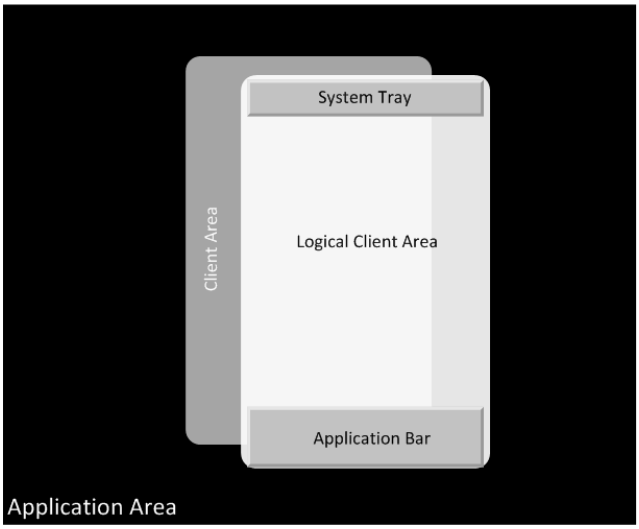
Each WP7 application has a single frame, and it includes areas for:

1. a page where application content is rendered. This is the content where widgets or graphics are rendered.
2. a reserved space for the system tray and application bar. It also exposes certain properties such as orientation to the application.



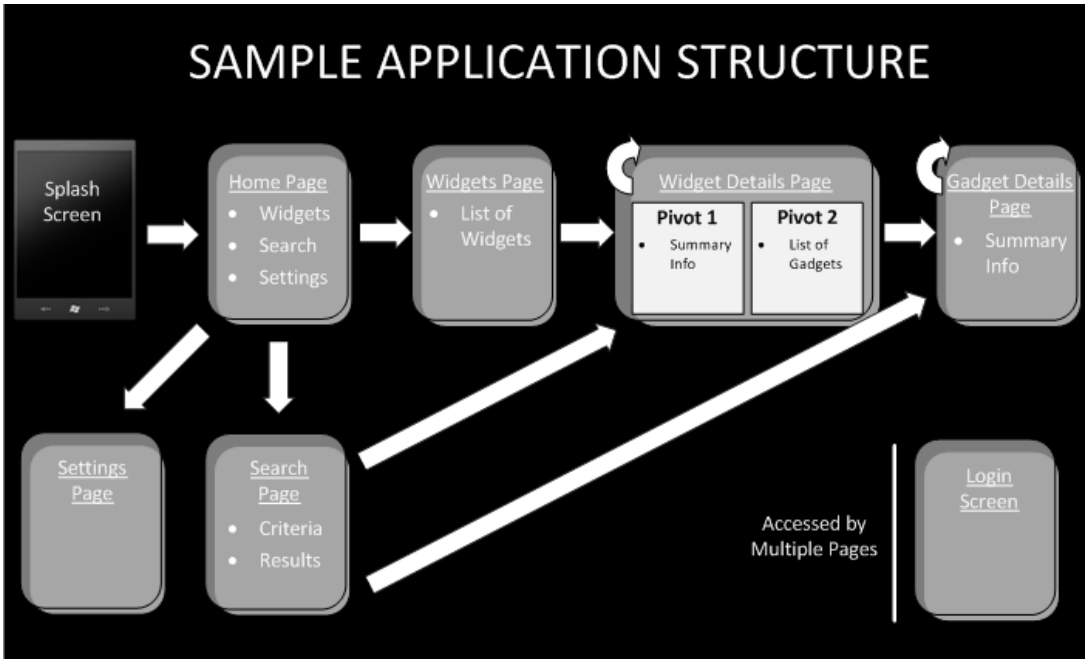

### System Tray and Application Bar

On WP7, the system tray includes indicators for various system-level status information. The application bar includes the area for the most common application menus, which may include various data views or tasks.




## Page Structure of WP7 Application

The following diagram shows the structure of a typical WP7 data-bound application, which resembles a navigation-based iPhone application.




When the user first starts the application, he or she would be presented with a splash screen, designed to welcome the user, as well as to create the perception of fast response. Splash screens are usually an image file, of the size of the display.

Usually the application starts with the home page; the main navigation page, with links for search, and other page widgets. Consider an application that shows information about the baseball teams and their players. The primary content page, marked “widgets” page above, will have the content of interest; e.g., a list of all baseball teams. In many cases, the home page will also be the primary content page.

The user can click on one of the team links to visit the team details page (“widget details page”) which can provide multiple views. The page may employ a pivot control or panorama to display different views such as the team summary and the list of all players (“list of gadgets”) from that team. Selecting one of the baseball players will take the user to the page with player statistics (“Gadget Details page”). Such a page may use controls such as textblocks,  multi-scale images, or other multimedia using a MediaElement control.

Users may also use the search widget to search and directly access the team page (“widget details”) or the player page (“gadget details”)

## Application Templates

As you know, XCode provides different templates for various iPhone applications. The following table shows the mapping between XCode application types and Visual Studio application templates.

| | | |
|---|---|---|
| Navigation-based | For information drilldown apps | Databound application |
| View based | For utility apps e.g. bubble level | Windows Phone application |

| | | |
|---|---|---|
| OpenGL-ES based | For Games | WP7 Game (XNA) application |
| Window-based | Flexible template to design any application | Windows Phone application |

You can choose the Windows Phone application template to either create an application with functionality similar to the view-based or the window-based iPhone application type. Lastly, the XNA based games application template will give you functionality similar to the OpenGL-ES application.

## Summary

In this chapter, we looked at the WP7 user interface guidelines. We showed the parallels between the application design goals of the iPhone platform and the WP7 platform. When you plan your WP7 application, you should be able to leverage your existing work on iPhone applications.

Revisit the application interface design to make sure you are taking advantage of the WP7 metro design that uses Windows Phone 7 interface guidelines. You will find that the WP7 tools offer a large library of controls and gestures that have close counterparts on the iPhone. Investigate the use of innovative controls like panorama, and explore the use of live tiles to build an engaging WP7 experience.

## Related Resources

To go deeper into the topic discussed, check:

1. Windows Phone 7 User Interface Guidelines
2. Windows Phone 7 Developer Tools
3. Silverlight for Windows Phone toolkit on CodePlex
4. Design resources for Windows Phone

Other Resources you may find useful:

1. Application Page Model for Windows Phone 7
2. Frame and Page Navigation Overview for Windows Phone

# Chapter 3: Developer and designer tools introduced to iPhone application developers

## Introduction

With the release of Windows Phone 7 developer tools, Microsoft brings the user-friendly, high productivity Visual Studio Development environment to Windows Phone 7. Developers who have used Visual Studio will find a familiar environment. Even iPhone application developers familiar with XCode will find it easy to migrate to WP7 developer tools and become productive quickly.

## Comparing the iPhone and Windows Phone 7 Tool Set

Visual Studio 2010 Express for Windows Phone is a full featured IDE specifically created for designing, developing and debugging Windows Phone 7 applications. This IDE, along with other tools, namely, Expression Blend, XNA Game Studio, and Windows Phone Emulator cover the entire cycle of Windows Phone application development.

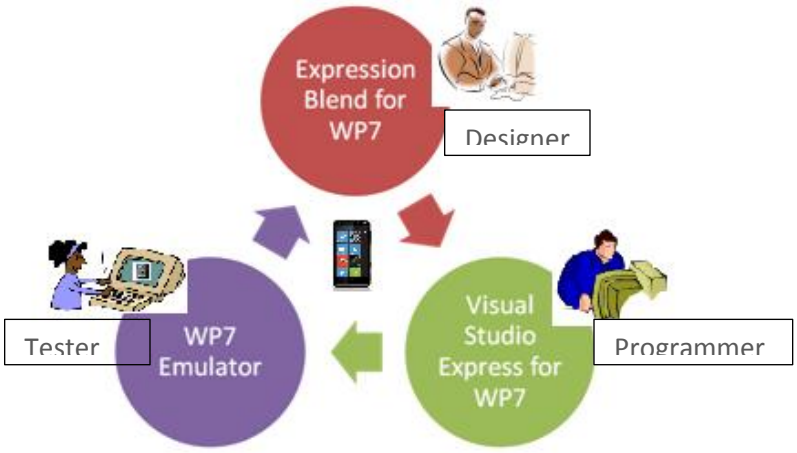
WP7 developer tools cover the full functionality that is provided by the iPhone application developer tools. The following table gives an overview of the functionality of each of these tools and how they correspond to iPhone development equivalents.

| Functionality | Audience | iPhone Development tools | WP7 Development tools |
|---|---|---|---|
| Primary UI design – colors, gradients, animation | UI Designer | 3rd party tools. | Expression Blend for Windows Phone |
| UI design | UI Designer / Programmer | Interface Builder | Visual Studio 2010 Express for Windows Phone and Expression Blend for Windows Phone |
| App development (Coding) | Programmer | XCode | Visual Studio 2010 Express for Windows Phone |
| Game development (Coding) | Programmer | XCode | XNA Game Studio |
| Testing / Emulation | Tester | iPhone simulator | Windows Phone Emulator (included in Visual Studio 2010 Express) |

As you plan to develop applications for WP7, your iPhone team structure and overall development process can remain the same. The entire team of designers, developers and testers, familiar with iPhone development tools, will find it easy to migrate to the WP7 toolset.

## Development Lifecycle and Window Phone 7 Developer Tools

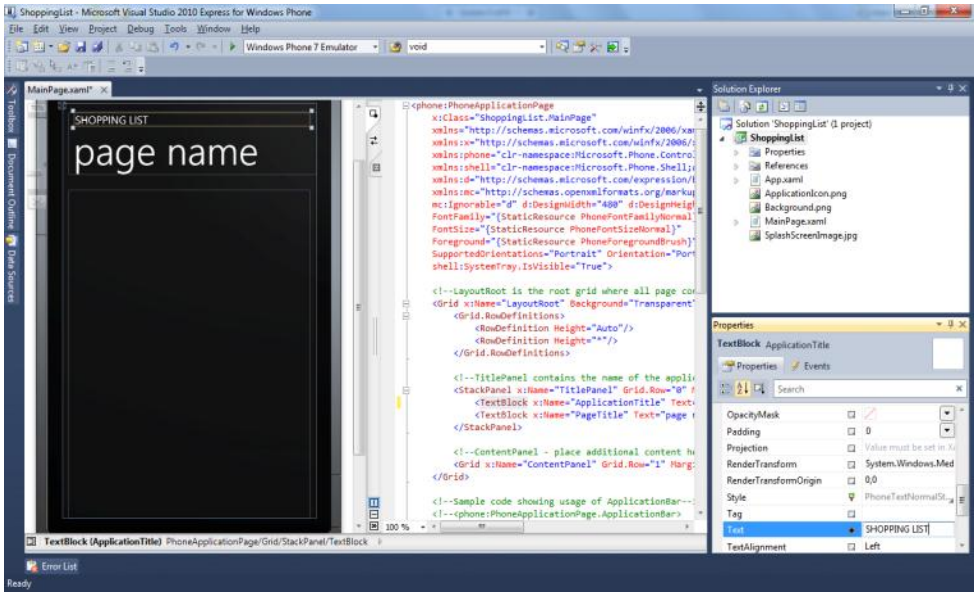
Windows Phone 7 development tools facilitate a close collaboration between designers and developers through the use of Expression Blend and Visual Studio. These two tools share the same file structure as well as actual source files. Expression Blend uses XAML for UI design, a declarative XML based language, which is also consumed by Visual Studio. This allows the designer and the developer to work seamlessly together while it provides clear separation of responsibilities between the two.




## Project Management

Like XCode, Visual Studio Express for WP7 is a full featured IDE. It allows developers to manage the entire structure of the development project; the source files as well as the various resource files. Visual Studio allows you to configure the application codebase, called a Visual Studio Solution, as a collection of projects, i.e. as a separate functional unit. This makes it easy to manage source files, to share code as well as to manage the work among team members. Visual Studio integrates a compiler and a debugger, both of which can be invoked either interactively or via the command line.

Let us create a sample application. Start Visual Studio Express for WP7 and click **File**, then **New Project.** In the **New Project** dialog select **Windows Phone Application.** Type "ShoppingList" for the name of the project and click **OK**. Visual Studio will create a new project for you as shown below. The Solution Explorer window shows the solution we just created. This solution has only one project, also named ShoppingList. The project contains the sources, resources and properties.




Unlike XCode, Visual Studio Express for WP7 does not provide integration with source control. You can use Visual Studio Professional edition which integrates various source control systems, such as Subversion, that iPhone application developers are familiar with. Alternatively, you can use the Visual Studio Team System, an edition designed particularly for greater communication and collaboration among software development teams, for developing your WP7 application.

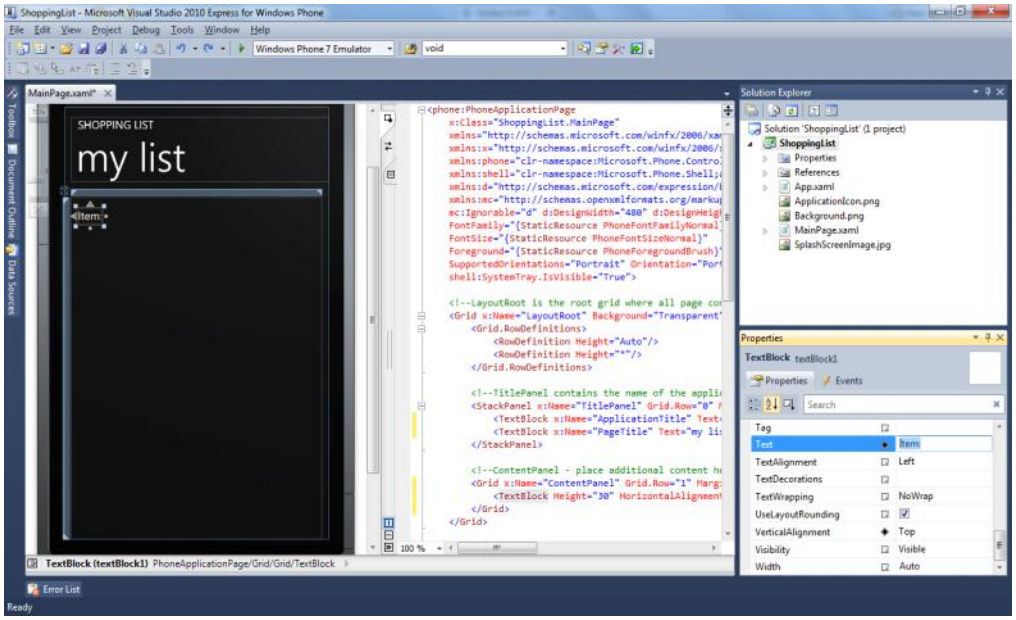
## UI Design Tools

WP7 developer tools include two UI design tools, namely, Expression Blend and Visual Studio UI designer. WP7 uses SilverLight, with its XML markup language, for the UI specification.

Visual Studio UI design tool is comparable to Interface Builder. iPhone application developers who know Interface Builder will find it easy to use this tool. The main page for our ShoppingList solution, MainPage.xaml, is already opened in the VS UI designer tool for editing (shown above).

Let us change the title of the application, as well as the title of the current page. Right click on the title, “MY APPLICATION” and select **Properties**. In the properties window, select **Text** and type “SHOPPING LIST.” Similarly, change the title of the page by typing “my list” in the **Text** property of the title.

Open the **Toolbox,** drag a **TextBlock** and drop it on the page. Position it so that it is at the top left. Right click on the **TextBlock** and update its Text property to “Item:”

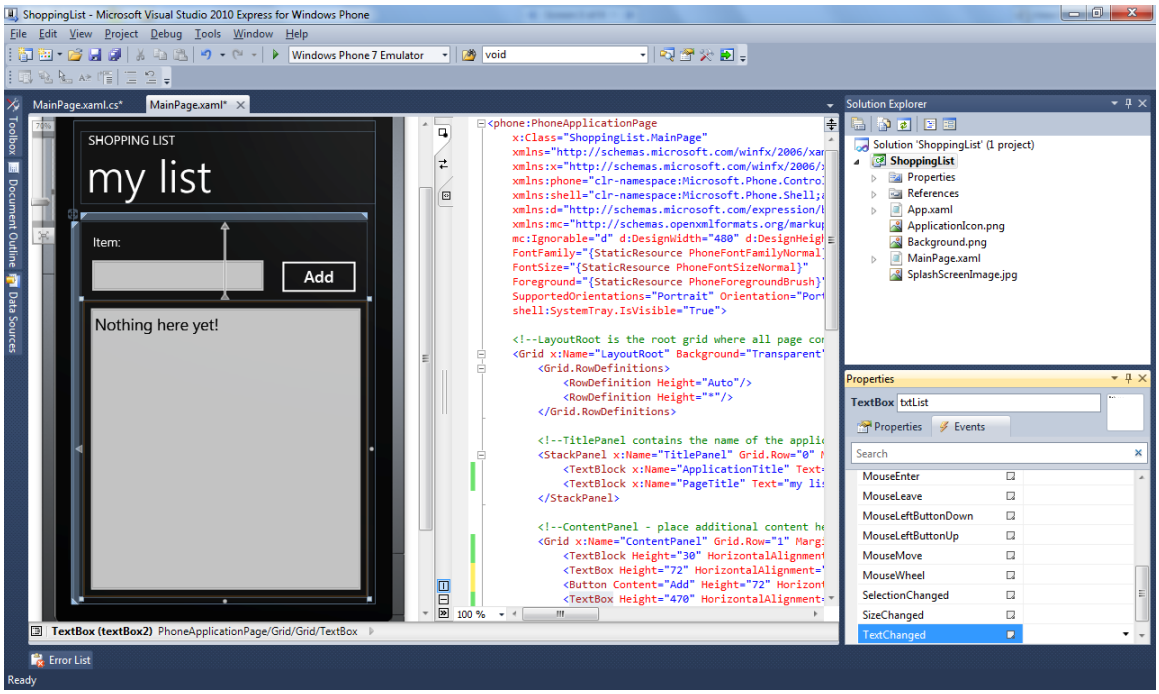



Drag a **TextBox** from the **toolbox** and place it underneath the above the textblock. Update its **Text** property to wipe it clean. Right underneath Properties, click on "TextBox1", and type "txtItem" to change the ID of the textbox to txtItem. Resize the textbox by dragging its right bottom corner so that its width is 300.

Similarly, drag a button and drop it to the right of the TextBox. Change its **Content** property to "Add", and its **ID** to "btnAdd". Resize the button so that its width is 140. And finally, drag another TextBox and place it underneath the txtItem textbox. Resize it so that it covers the rest of the phone screen. Update its **ID** to "txtList" Update its **Text** property to "Nothing here yet!" Your application should look something like this:



Windows Phone

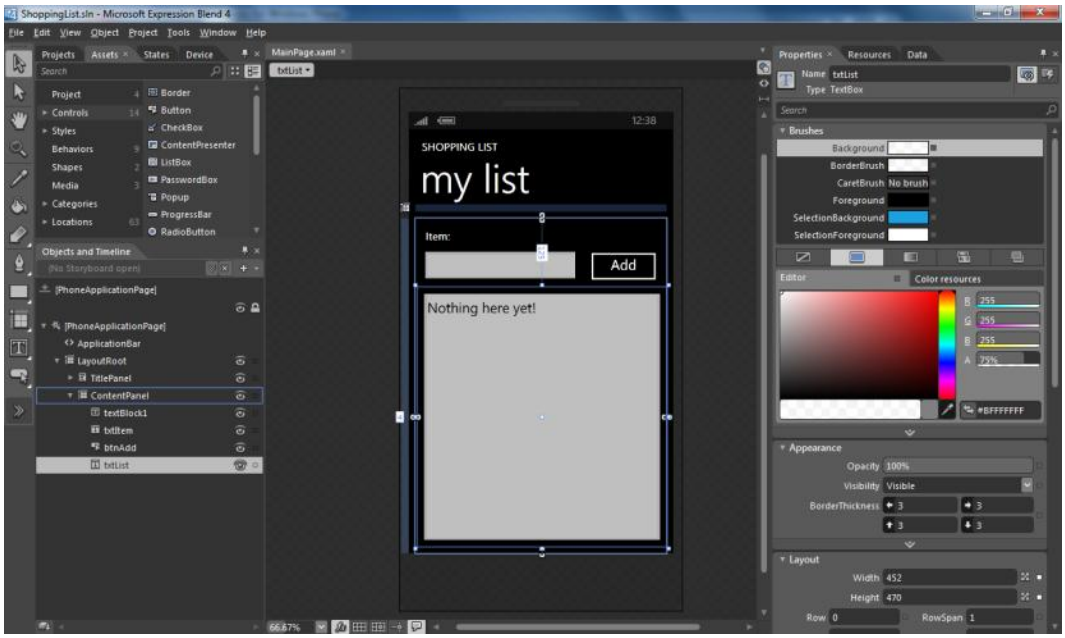



Click **F5,** or **Debug** and **Start Debugging,** to compile the application and launch it. This will start the WP7 emulator, deploy the ShoppingList application and run it. You can click on **Add,** but nothing will happen as we have not written any logic yet.

Developers can use context menus to add event handlers or set control properties. Its integration with Visual Studio allows for direct manipulation of controls and makes it easy to add logic to UI controls.

## Expression Blend for WP7

Expression Blend for WP7 is a full featured visual UI design tool created for designers. There is no exact counterpart to this in the iPhone development toolset. Similar to VS Design tool, Expression Blend also allows drag and drop to design the UI. The tool, shown below, allows pixel accurate layout of controls. They can easily create and use color palettes and gradients, as well as special effects such as reflections and shadows. The tool can import Photoshop files, to make it easy to bring your iPhone application resources to your Windows Phone application. Designers can also use the tool to define application behavior, as well as certain animations, without any programming.

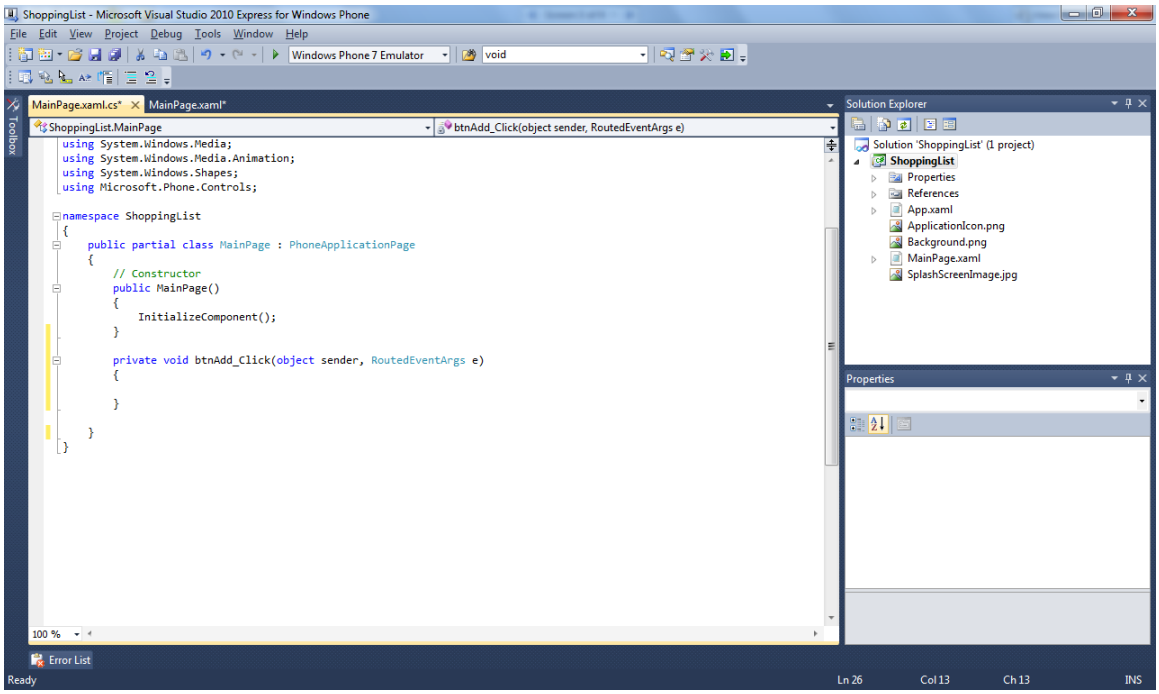



While designers use Expression Blend, and programmers use the Visual Studio Design tool to hook up their application logic to the UI design, the VS UI design tool can also be used for the UI design, as we saw earlier. Both tools include the same control set, that provides accurate fidelity to their run time visual representation, making it easy to visualize the application. The two design tools use the same project structure and share source files. Both tools consume/produce XAML, the Silverlight XML declarative markup language, for the interface design. This makes it very convenient for a designer to work on the design using Expression Blend while the developer uses Visual Studio to design the logic behind the application. It creates a smooth design and development workflow.

## Editing code

Visual Studio includes a simple to use, full featured, yet configurable, source editor. It provides various features that will be familiar to XCode users. These include flexible search, rich editing, code formatting, and the ability to outline/hide code.

Let us add some logic to our application. Stop the running application by clicking **Debug,** followed by **Stop Debugging**. Double click the "Add" button which will open MainPage.xaml.cs with a method **btnAdd_click** in the **MainPage** class.



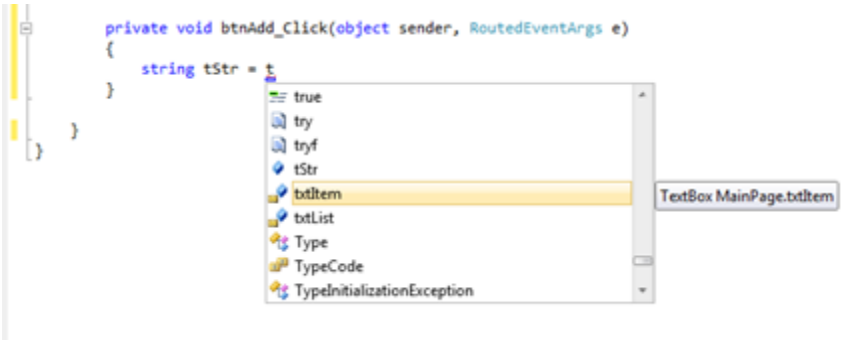

Edit the newly added method to add logic to add items to the shopping list. Type:

```
string tStr = txtItem.Text;
```

As soon as you type "t" for txtItem, VS will bring up the auto-completion dialog as shown below. The Visual Studio counterpart for XCode auto-completion is called IntelliSense.



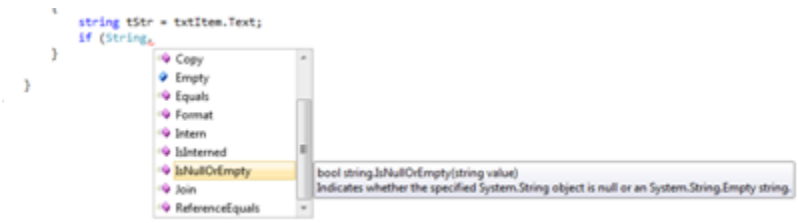
Windows Phone



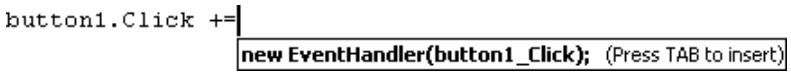

Also type:

```
if (!String.IsNullOrEmpty(tStr))
```

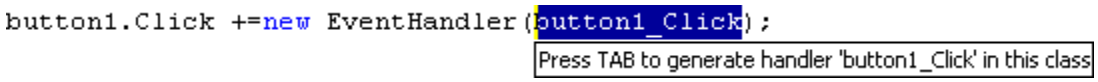
As soon as you type, “String.” VS will pop up the auto-completion dialog. Typing “Is” will take you to the class methods of the String class.




VS IntelliSense is richly featured. As opposed to relying on the history alone, it disambiguates using the code context and .NET reflection, for intelligent auto-completion. It can suggest or even complete variable names, parameters, as well as class and method names. It even generates appropriate code where needed, as shown below using an unrelated code fragment:




To complete the event hookup, it will also generate an empty stub for the event handler, i.e., the button1_click method.

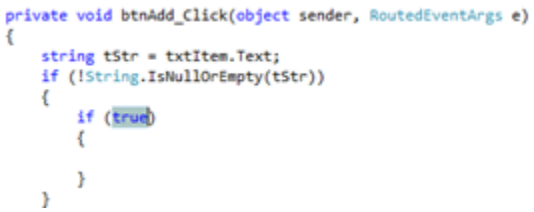



Visual Studio provides another very useful feature called Code Snippets, which is a counterpart to text macros in XCode. It allows you to insert code fragments in the active file with a few mouse clicks. Visual

Studio ships with a large number of snippets and developers can create their own library of snippets. They can also be indexed and searched using user defined terms.

Type **ctrl+k ctrl+x** to bring up the **Insert Snippet** prompt. Select Visual C#, followed by "i" to select a code snippet for "if statement", which will insert an if statement in the code.


The inserted snippet identifies the parts the user needs to complete:




Type the remaining code, so that the body of the method is as follows:

```
string tStr = txtItem.Text;
if (!String.IsNullOrEmpty(tStr))
{
    if (txtList.Text == "Nothing here yet")
    {
        txtList.Text = "";
    }
    txtList.Text += txtItem.Text + "\n";
    txtItem.Text = "";
}
```

Visual Studio supports various refactoring mechanisms. Select any piece of code and right click to access the refactoring menu.

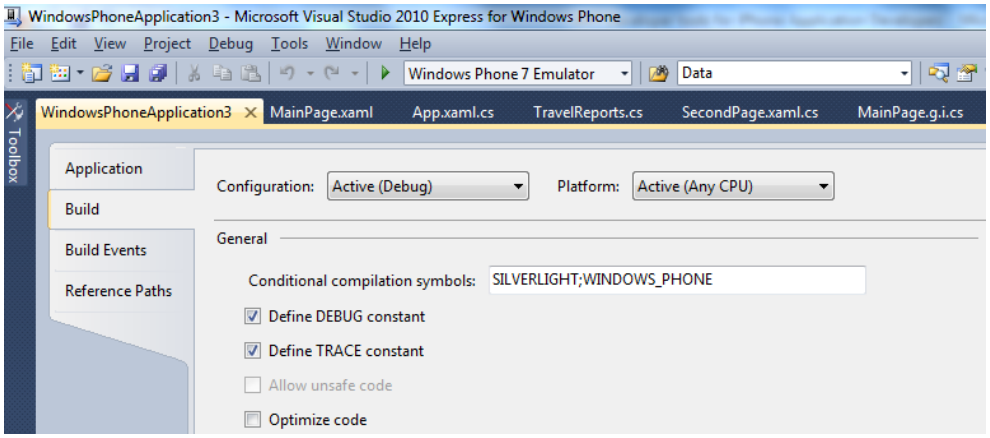
The Visual Studio editor is highly customizable. Developers can easily define different keyboard shortcuts or create their own macros. Macros help you automate repetitive actions by combining a series of commands and instructions together, making it easy to invoke them as one command. iPhone

application developers can easily customize the editor to use any shortcuts and keyboard combinations that they are familiar with. Instead of spawning a separate window for each file, as in XCode, the default view in VS uses tabbed windows. Developers can change this behavior to suit their need. They can change the way in which various windows are docked within the Visual Studio Shell.

## Building Applications

Similar to XCode, Visual Studio Express for WP7 allows you to build the Visual Studio solution on demand. Further, each project that is part of the solution can be built separately.

Visual Studio uses an XML based, declarative build system called MSBuild which can be compared with Ant/Nant. Builds can be invoked interactively or via a command line for batch processing. This system is flexible and allows you to build a specific target either as a debug build or as a release build.




### Emulator

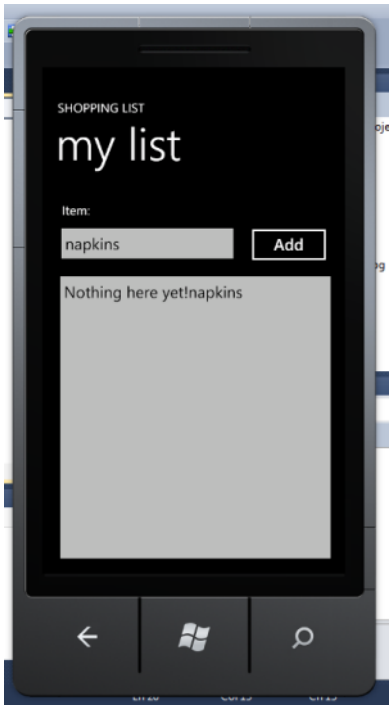
WP7 developer tools include an emulator that can be used effectively for testing applications. It provides features that are comparable to the iPhone simulator included in the iPhone developer tools.

The WP7 emulator provides a virtualized environment in which you can deploy, debug and test applications. The Windows Phone Emulator is designed to provide comparable performance to an actual device and meets the peripheral specifications required for application development. It can be invoked from Visual Studio to load an application package [.xap] within the emulator.

### Debugging

Visual Studio Express Phone 7 includes a very powerful symbolic debugger that can be used with the WP7 emulator or with a remote device. Once the application breaks into the debugger, the developer can view the variables in the application and control the execution.

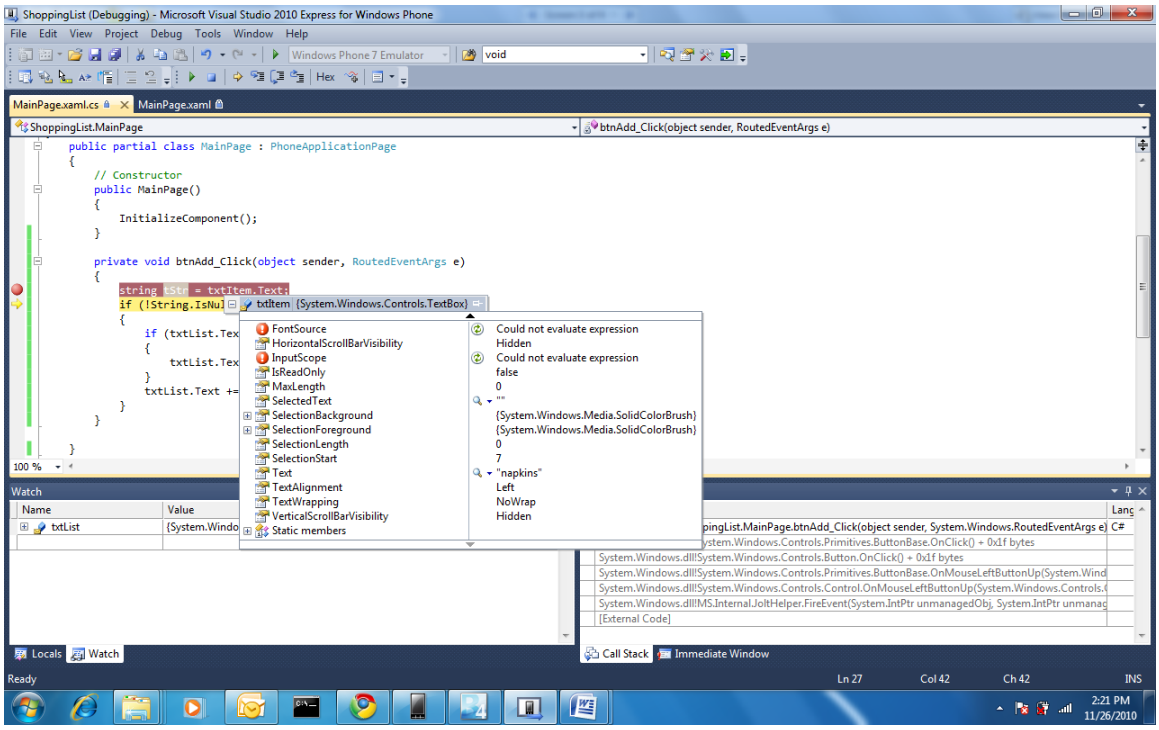
Let us look at the debugger in action. Press F5 to launch the application again. Type “napkins” in the textbox and click **Add**.




“Napkins” is added at the end of “Nothing here yet!” - not something we expected. In Visual Studio, click in the light blue area to the left of the “string tStr = txtItem.Text;” line in the code window. This will insert a breakpoint at that line.

```
private void btnAdd_Click(object sender, RoutedEventArgs e)
{
    string tStr = txtItem.Text;
    if (!String.IsNullOrEmpty(tStr))
    {
        if (txtList.Text == "Nothing here yet")
        {
            txtList.Text = "";
        }
        txtList.Text += txtItem.Text + "\n";
    }
}
```

Launch the application again using **F5.** When the application breaks into the debugger, hover over txtItem in the code and click “**+**” in the popup to view the variable txtItem, as shown below. The developer can view the variable, its type, its fields and properties. The picture below shows how you can walk up and down the type hierarchy to inspect the objects.




You can set a "watch" on certain variables to inspect them continuously. Right click txtList, followed by **Add Watch.** The watch window will show the variable txtList. Expand txtList by clicking on "**+**".


Step through the code using **F10** to see that control does not enter the if statement.

```
if (txtList.Text == "Nothing here yet")
 {
     txtList.Text = "";
 }
```

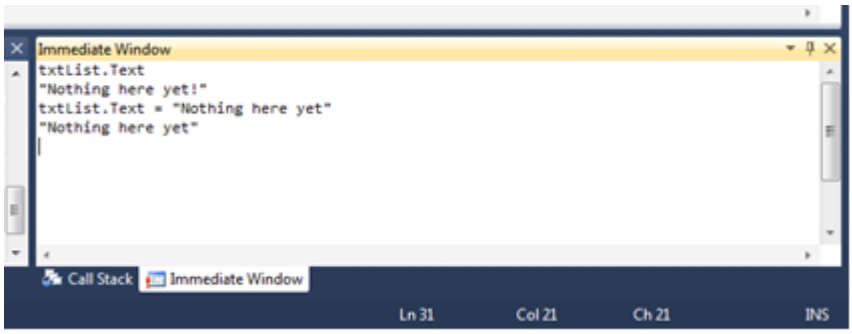
Observe in the watch window that the value of txtList.Text is "Nothing here yet!", whereas it is getting compared with "Nothing here yet" (with no exclamation point.) Therein is our bug! Change that statement to add the exclamation point, as follows:

```
if (txtList.Text == "Nothing here yet!")
```



Windows Phone

While in the debugger, the developer can use the VS ‘immediate mode’ where one can write managed code instructions to modify or view the variables or execute some code to help with debugging.




Update the code and relaunch the application. Test it by adding couple of items to the shopping list.


Overall, you will find that, with the power of the managed programming environment, debugging a WP7 application is very easy. Unlike an XCode application, where you have access to assembly instructions, memory dumps and various registers, the WP7 application debugging is done entirely at the application level, using C# code and types.

In addition to the above debug facilities, the .NET framework includes two specific classes, Debug and Trace, that make it easy to write run-time debug messages to the output window. C# also supports an assert statement, which is evaluated at run time. If the statement evaluates to true, nothing happens, but if the statement returns false, the program breaks into a debugger.

## Summary

The Windows Phone 7 developer toolset includes rich tools designed to support every step in the entire application development lifecycle. The design, development and testing tools are amenable to existing iPhone team roles and processes.  The tight integration between theWP7 tools can help you streamline your design, development and testing workflow.  These tools provide end-to-end functionality and are highly customizable, with the power to make your team quickly productive.

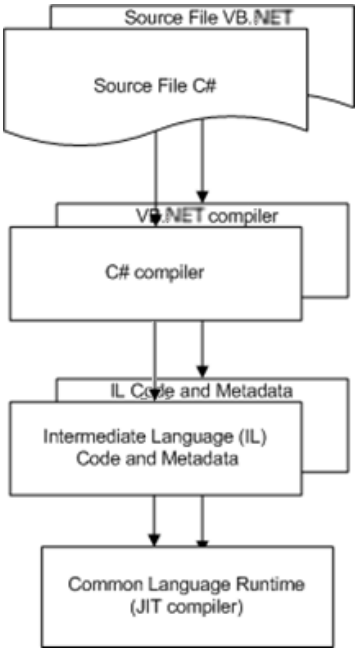
# Chapter 4: C# programming introduced to Objective-C programmers

In the previous chapter, we looked at the user interface guidelines for WP7 applications. We will now dive deeper into what it takes to implement a WP7 application.

In this chapter, we will look at the various C# features that map to the most common Objective-C features. We will provide code snippets which will ease your way into C# code. We will point to the key C# features that help you write safe code and enhance productivity.

## Introduction to Managed Programming

WP7 only supports managed programming in C# or VB.NET. Before we jump into the details of C#, let us briefly review managed programming.




The C# compiler (and similarly, the VB compiler) compiles the C# (or VB.NET) code into an intermediate language (IL) bytecode and metadata. The Common Language Runtime (CLR) executes the byte code. It uses metadata to manage type safety, exception handling, array bounds, etc. The CLR also manages memory and performs garbage collection. In contrast, Objective-C code is compiled into ARM binary code and executed directly.

## Comparison between C# Features and Objective-C Classes

## Class Declaration

Let us start with an example program. In contrast to Objective-C, C# does not separate the class definition and the implementation. The compiler derives the metadata about the classes from the class implementation itself. You will also notice that you do not need to define each class in a separate file as in Objective-C.

In the example, the public signature of the class Person consists of just the property, age, and the constructor. The rest of the class implementation is opaque.

```
using System;                           // C# does not import a .h file,
uses metadata
namespace FirstApplication              // scope for classes. No Obj-c
counterpart
{
    class Person              // only uses class implementation
    {
        private DateTime birthDate;    // a private field accessible to
this class
        private int ageOn(DateTime date)    // a private method
        {
            TimeSpan span = date.Subtract(birthDate);  //uses a
.notation to invoke
            return span.Days;
        }
        public int age                  // this is a property.
        {
            Get                         // just a getter; it's a read-
only property
            {
                return this.ageOn(DateTime.Now);
            }
        }
        public Person( DateTime dob)  // instance constructor. Unlike
Objective-C
        {                   // it combines allocation and
initialization
            birthDate = dob;
        }
    }
    class Program                       //Unlike Obj-C, another class in
the same file.
    {
        static void Main(string[] args) // main entry point into the
program
        {
            Person p = new Person(new DateTime(1973,11,12));
```

```
//construct an instance
            System.Console.WriteLine("The age is  is" +
p.age.ToString());
            DateTime dt = p.birthDate; //error in compilation
birthDate is private
        }
    }
}
```

Instead of using the `import` statement, C# employs a `using` statement to refer to the metadata of other classes. The `namespace` declaration, shown at the top of the file, is used to both declare scope and organize the code. You can access classes in other namespaces by referring to a fully qualified name. See the reference to System.Console.WriteLine in the example above, where console is in the System namespace.

Objective-C uses a message passing syntax consisting of square brackets, and a dot-notation for accessing properties. C# uniformly uses the "." notation for referring to all methods, fields and properties.

## Strong Typing

In contrast to Objective-C, C# is a very strongly typed language. Types must be specified for variables as well as input/output parameters. Types are enforced strictly by the compiler. Objective-C uses weak typing for collection classes such as NSArray and NSDictionary. In the section on generics below, we will see how C# uses strong typing for collection classes.

```
int a = 5;
int b = a + 2; //OK
bool test = true; // OK
int c = a + test; // Error. Operator '+' cannot mix type 'int' and
'bool'.
```

The example above shows the strong typing for primitive types. Strong typing works similarly for all classes.

## Class Constructors

In contrast to the separate alloc and init statements of Objective-C, in C#, instance constructors are used to create and initialize instances. For example, p, an instance of the Person class, can be both constructed and initialized with a given birthdate, in a single statement.

```
Person p = new Person(new DateTime(1973,11,12));
```

### Properties

Developers often need to decide about whether to implement a member as a property or a method. In this case, the design pattern is identical for Objective-C and C#. In general, the guidance is to use properties when accessing data, and to use methods when there is an action taken on the data.

As opposed to the Objective-C @property attribute, C# properties are declared by the explicit definition of a getter, a setter, or both. You can make the property read-only by providing just the getter, write-only by providing just the setter or read-write, by providing both.

### Parameter Types

Similarly to Objective-C, C# uses value parameters by default. While C# does not have pointers, it allows passing of parameters by reference by using the 'ref' modifier. Instead of pointers, parameters with `ref` can be used where you want to achieve side effects in a method. In some cases, reference parameters are more efficient, since they avoid data copying.

```
void Foo (ref int x, int y)
{
          x = 0;
          y = 0;
}
..
int a = 5;
int b = 8;
Foo (ref a, b);
//a is zero and b is still 8
```

C# also provides parameters with an `out` modifier which denotes parameters that must be initialized by the called method before returning. This design pattern is often used to return the error in addition to the value of the function.

### Access Privileges

In Objective-C, access privilege can only be specified on variables. Methods which are present only in the .m file are private. On the other hand, C# allows access privileges on fields (e.g., birthDate), properties (e.g., age) and methods (e.g., ageOn). It uses `public, private` and `protected` as modifiers to denote three different levels of access privileges.

In the above example, the compiler will error out on p.birthDate since that variable is private and therefore is not accessible from the Program class. Similarly, the method ageOn is also private and inaccessible from the Program class.

## Methods with multiple parameters

Both Objective-C and C# support methods with multiple parameters. In Objective-C method parameters are positional and named, i.e., the names of formal parameters are used while passing actual

parameters. The name of the method is comprised of everything to the left of the colon (":"), for example, the name of the Objective-C method below is addEmployee:name:age:. While C# traditionally used positional and unnamed parameters, the latest version of C# has also introduced named parameters. The following example shows the comparative syntax for Objective-C and C#.

| Objective-C | C# |
|---|---|
| - (void) addEmployee:(NSString *)name id:(int)id age:(int)age | void addEmployee(string name, int id, int age); |
| [off addEmployee:@"Phil" id:2345 age:23]; | Off.addEmployee("Phil",2345, 23);<br><br>Off.addEmployee(name: "Phil", age:23, id:2345); |

Objective-C does not support method overloading. While it does not allow exactly the same method signature with different parameter types, the following design pattern is commonly used in Objetive-C programs:

```
- (void)insert:(myClass *)obj atIndex:(NSInteger)index
- (void)insert:(myClass *)obj beforeObj:(myClass *)obj

[mylist insert:obj1 atIndex:4];
[mylist insert:obj2 beforeObj:obj1];
```

As we saw earlier, the names of these two methods are different and are "insert:atIndex" and "insert:beforeObj" respectively.

On the other hand, C# explicitly supports method overloading. Using information about the parameter types, C# disambiguates between methods with the same name.

```
void insert(myClass obj, int index);
void insert(myClass obj, myClass before);
```

The method insert may be called with both signatures:

```
list.insert(myObj1, 4);
list.insert(myObj1, myObj2);
```

Now that we have examined some of the basic class concepts in C#, let us look at another example:

```
using System;
namespace SecondApplication
{
    struct Point                          // In contrast to Obj-C, C# structs are closer
    {                                     // classes.
```

```
    public double x;                    // struct fields can also have access modifiers
    public double y;
    public Point(double p1, double p2)  //a constructor for the struct
    {
        x = p1;
        y = p2;
    }
}
interface IThreeDShape                  // an interface, like an Objective-C protocol only
{                                       // defines the behavior
    double volume
    {
        get;                            // Volume is a read-only property. no setter
    }
}
abstract class Shape                    // this class is marked abstract, i.e. may not be instantiated.
{
    protected Point origin;             //only derived classes may access
    protected static int counter = 0;   // Similar to class variables in Obj-C
    public string ID;
    protected Shape()                   //a constructor. Same name as the class name
    {
        counter++;                      // class variable being updated
    }
    public Point Origin                 // similar to objective-C property
    {
        set
        {
            origin = value;
        }
    }
    public abstract double Area         //denotes that this property must be overridden
    {                                   // in a derived class
        get;
    }
    public abstract bool contains(Point p);  // this method must also be overridden
}

class Rectangle : Shape                 //Similar to obj-c, single iinheritance
{
    public double length;               //field accessible from others
    public double width;
    public Rectangle(Point o, double l, double w)   //a public constructor
    {
        ID = "Rectangle_" + counter.ToString();
        origin = o;
        length = l; width = w;
    }
    public Rectangle(double l, double w)  // one constructor using another constructor
                                          //creates a rectangle at the origin
        : this(new Point(0, 0), l, w)
    {
    }
    public override double Area         // unlike Obj-C, overridden method must
    {                                   // use override keyword
        get
        {
            return length * width;
        }
    }
    public override bool contains(Point p)
    {
        if ((origin.x < p.x && origin.x + length > p.x) || (origin.x > p.x && origin.x - length < p.x))
            if ((origin.y < p.y && origin.y + length > p.y) || (origin.y > p.y && origin.y - length < p.y))
                return true;
        return false;
    }
}
class Square : Rectangle
{
    public double side;
    public Square(double s)
        : base(s, s)  //constructor
    {
        ID = "Square_" + counter.ToString();
        side = s;
    }
```

```
    }
    class Cube : Shape, IThreeDShape          //similar to obj-c, class implements interface (protocol)
    {
        public double side;
        public Cube(double s)
        {
            ID = "Cube_" + counter.ToString();
            side = s;
        }
        public override double Area
        {
            get
            {
                return 6 * side * side;
            }
        }
        public double volume
        {
            get
            {
                return side * side * side;
            }
        }
        public override bool contains(Point p)
          …
    }
    class SecondProgram
    {
        static void printVolume(IThreeDShape tdShape)
        {
            Console.WriteLine("The volume is " + tdShape.volume);
        }
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle(5.0, 3.0);
            Cube c = new Cube(4.0);
            SecondProgram.printVolume(c);
            double a = r.Area;
            Console.WriteLine("The area of rectangle " + r.ID + " is " + a.ToString());
            bool b = r.contains(new Point(1, 2));
            Console.WriteLine("The point is in " + b.ToString());    // will print TRUE
        }
    }
}
```

## Inheritance

Like Objective-C, C# also uses a single inheritance mechanism. Inheritance is specified by listing the parent class after the name of the class as shown below. In the above example, the class Rectangle inherits from the class Shape, whereas the class Square inherits from the class Rectangle.

```
class Rectangle : Shape
class Square : Rectangle
```

In C#, the constructor of the base class is automatically invoked when constructing an instance of a derived class. However, a derived class can invoke a specific constructor of the base class if needed as shown in the constructor of the Square class.

```
public Square(double s): base(s, s)  //constructor. Calls parent constructor explicitly
```

In contrast to Objective-C, a C# derived class may not override a method by just redefining it. The class must use the keyword “override” in its method definition.

```
public override bool contains(Point p)
{
        …
}
```

## Protected Access

Objective-C provides protected variables, but methods cannot be protected. In C#, access to fields, properties and methods can also be controlled using the protected modifier.  You can implement protected variables in C# by using the protected access modifier, as shown below:

```
protected Point origin;
        protected static int counter=0;
```

## Instance vs Class Level Access

While Objective-C and C# use different syntactic notation for static methods or variables, they behave the same way.  C# uses a ‘static’ modifier to denote class level methods, fields or properties. Everything else is at an instance level. In the above example, counter is a class level variable.

```
protected static int counter=0;
```

## Abstract Classes

Abstract classes, are classes that cannot be instantiated. While Objective-C does not provide a syntax for abstract classes, many programmers use them by returning NULL from the abstract class init method. The class Shape, defined above in C#, is an abstract class and requires that both Area and the method contains must be overridden in any derived classes.

```
abstract class Shape
    {
        public abstract double Area
        {
            get;
        }
        public abstract bool contains(Point p);
```

## Interfaces

Objective-C protocols and C# interfaces are similar. In the example below, IThreeDShape defines an interface that is implemented by the Cube class.

```
interface IThreeDShape
    {
        double volume
        {

          ...
    class Cube : Shape, IThreeDShape
```

## Polymorphism

Polymorphism works the same way in both Objective-C and C#. A C# derived class can be passed as a parameter to a method that expects a base class. Similarly, a class that implements a particular interface can also be passed as a parameter to the method. This is shown in the example below, where an object of the class Cube is passed as a parameter, where the method expects an object of the class IThreeDShape.

```
static void printVolume(IThreeDShape tdShape)
        {
            Console.WriteLine("The volume is " + tdShape.volume);
        }
         …
        Cube c = new Cube(4.0);
        SecondProgram.printVolume(c);
```

## Structs

In contrast to the C-based structs used in Objective-C, C# structs are closer to classes. C# structs can have constructors, methods and properties as well as access modifiers. However, the primary difference between a struct and a class is that a struct is a value type, versus a class, which is a reference type.

```
struct Point
    {
        public double x;
        public double y;
    …
```

## Object Lifecycle – Creation and Deletion of Objects

Memory management is very different in Objective-C and C#. In contrast to Objective-C, C# performs automatic memory management. As we saw earlier, developers do not allocate memory, but use the "new" operator to create objects on the heap and initialize them. Equally important, in C#, the developer is not responsible for tracking memory usage or knowing when to free memory. When the object is no longer accessed by the code, the object is eligible for garbage collection. Periodically, the .NET CLR garbage collector frees up the memory for such objects.

In rare circumstances, developers may need to perform cleanup at the time the object is destroyed. C# allows the use of destructors, but in practice this is rare.

## Other Topics

**Type Checking v/s Reflection**

In Objective-C, you can check the type of the class or determine if an object supports a particular method and invoke the method on that object. In C#, reflection is a versatile feature. You can use reflection to get the type information from an existing object, dynamically create an instance of a type, bind the type to an existing object, invoke its methods or access its fields and properties.

The following table explains the mapping between dynamic type checking in Objective-C and the corresponding C# reflection features.

| Objective-C Dynamic Type Checking | Explanation | C# Reflection |
|---|---|---|
| isKindOfClass: classObj | Is Object a subclass or member | type.IsSubclassOf(typeof(BaseClass)) |
| isMemberOfClass: classObj | Is Object a member of | object.getType() or typeof |
| respondsToSelector: selector | Does the object implement the method | type.GetMethod(MethodName) |
| instancesRespondToSelector: selector | Does the class respond to the method | type.GetMethod(MethodName) |
| performSelector: selector | Invoke a method | type.InvokeMember(…) |

## Exception Handling

Exception handling is similar in C# and Objective-C. You use a try-catch block to handle exceptions. Additionally, you can either catch specific exceptions or use a catch-all statement. This is similar to @try, @catch and @finally statements in Objective-C.

```
try
            {
                //block of code
            }
            // Most specific:
            catch (ArgumentNullException e)
            {
                Console.WriteLine("{0} First exception caught.", e);
            }
            // Least specific:
            catch (Exception e)
            {
                Console.WriteLine("{0} Second exception caught.", e);
            }
```

## Key class libraries compared

### Strings

C# provides a very comprehensive string class, which gives you all the features that you are familiar with in the NSString class.

| Objective-C Feature | C# | Notes |
|---|---|---|
| NSString | String greeting = “Hello WP7!”;<br>Int length = greeting.Length; | |
| Comparison | String color = “pink”;<br>If (color == “red”)<br>System.Console.WriteLine(“Matching colors!”);<br>string name = “Joe”;<br>if (string.compare(name, “Jack”) > 0)<br>System.Console.WriteLine(name + “ comes later”); | Strings can be compared using ==. They can be compared lexicographically using compare. |
| Concatenation | System.Console.WriteLine (greeting + " You rock!") | Strings can be concatenated with simple ‘+’ operator. (This is called operator overloading) |
| Splitting | string rainbow = "Violet, Indigo, Blue, Green, Yellow, Orange, Red";<br>string[] rainbowColors = rainbos.Split(',');<br>foreach (string color in rainbowColors)<br>System.Console.WriteLine (color); | |

## Arrays

| Objective-C Feature | C# | Notes |
|---|---|---|
| Arrays of primitive types such as int, float | int[] table;<br>table = new int[3];<br>string[] names = new string[3] {"Peter", "Paul", "Mary"}; | Array size is not part of the array declaration.<br>Arrays are explicitly initialized. |
| Multi-dim arrays of primitive types | Int[,] mAray;<br>Int[][] jaggedArray;<br>string[][] singers = {new string[] {"Peter", "Paul", "Mary"}, new string[]{“Paul”,“Art”}}; | C# supports jagged arrays, or arrays of arrays, and they need not be rectangular.<br>Note Arrays of strings, i.e. objects, work the same way. |
| NSArray - Immutable Arrays of objects | | There is no counterpart to immutable arrays in C#. |
| NSMutableArray Mutable array of objects | List<string> colors = new List<string>; //list of strings<br>Colors.Add(“Red”);<br>Colors.Add(“Green”);<br>Colors.Insert(1,”White”);<br>String myColor = Colors[0]; //”Red”<br>Colors[colors.IndexOf(“Red”)] = “Pink”; // replace Red with pink | You can use Lists as a replacement for mutable arrays.<br><br>You may also use ArrayLists. |

## Dictionaries

C# provides a generic dictionary class that provides the functionality of NSMutableDictionary. It allows addition, lookup and removal of objects in the dictionary. Since it uses generics, it also utilizes strong typing.

| Objective-C Feature | C# | Notes |
|---|---|---|
| NSDictionary - Immutable dictionary | | There is no counterpart to immutable dictionary in C#. |
| NSMutableDictionary - Mutable dictiojary of objects | Dictionary<string, int> d = new Dictionary<string, int>();<br><br>d.Add("Honda", 124);<br><br>d.Add("Toyota", 95);<br><br>d.Add("Ford", 135);<br><br><br>// See if Dictionary contains string<br><br>if (d.ContainsKey("Ford")) // True<br><br>{<br><br>int v = d["Ford"];<br><br>Console.WriteLine(v);<br><br>} | You can use Dictionary as a replacement for NSMutableDictionary. |

# New features of C#

## Generics

Generics introduce the notion of type parameters, that make it possible to design classes that are type safe, even though the actual type is deferred till the object instantiation. For example, here is how you define a generic stack:

```
Stack<int> intStack = new Stack<int>();          // intStack is a stack
of int
intStack.Push(1);                                // OK
intStack.Push(2);                                // OK
int number = intStack.Pop();                     // this is a type safe
assignment
Stack<string> strStack = new Stack<string>();    //the type of strStack
is different from type of intStack
strStack.Push("green");                          // OK
strStack.Push(23);                               // compiler error
```

The Stack<T> uses T as a type parameter allowing you to instantiate a stack of any type, e.g. Stack<int> or Stack<string> and use them in a type safe manner.

Use of generics is closest to the use of id in Objective-C collection classes such as NSDictionary.

## Operator Overloading

Operator overloading permits a user defined implementation of user-defined operators for user-defined classes. Consider the following example of a Complex number struct. Operator overloading allows you to define a '+' operation using a natural syntax.

```
public struct Complex
    {
        public int real;
        public int imaginary;
        // Declare which operator to overload (+), define how it is
computed
        public static Complex operator +(Complex c1, Complex c2)
        {
            return new Complex(c1.real + c2.real, c1.imaginary +
c2.imaginary);
        }
       …
    Complex c1 = new Complex(3.0, 4.0);
    Complex c2 = new Complex(4.0, 5.0);
    Complex cSum = c1 + c2;
```

## Delegates

Objective-C developers often use delegation for notification as to when an asynchronous operation is completed. In C#, delegates are similar to function pointers in C or Objective-C. In this design pattern, a class delegates another class, not known at compile time, to complete its action.

```
using System;
namespace DelegateExample
{
    public class ConsoleLogger
    {
        public static void WriteString(string s)
        {
            Console.WriteLine("Writing to console log: {0}", s);
        }
    }
    public class FileLogger
    {
        public static void LogString(string s)
        {
            Console.WriteLine("Logging to file log: {0}", s);
        }
    }
    public class DelegatesTest
    {
        public delegate void StringDelegate(string s);
        public static void Main()
        {
            StringDelegate Writer, Logger;              // define twp
StringDelegate objects
            Writer = new StringDelegate(ConsoleLogger.WriteString); //
Create delegates with appropriate methods
            Logger = new StringDelegate(FileLogger.LogString);
            Writer("Warning message 1\n");              // Send to
Console Writer delegate method
            Logger("Warning message 2\n");              // Send to File
Logger delegate method

            StringDelegate MultiLogger;                 // to act as
the multicast delegate
            MultiLogger = Writer + Logger;              // combine the
two delegates,
            MultiLogger("Warning message 3");           // This should
get sent to both delegates
        }
    }
}
```

Let us look at the above example, where StringDelegate is defined as a function that takes a string as a parameter and returns void. Three different delegates, writer, logger and multiLogger, are constructed by passing in methods that have the same signature as the StringDelegate declaration. This can be contrasted with Objective-C delegates, which are usually defined using protocol declaration.

Calling Writer invokes the writeString method of ConsoleLogger to print the message to the console. On the other hand, calling Logger invokes the logString method of FileLogger to log the message to the file. As you can see, delegates achieve indirection while providing type safety. Delegates may be concatenated, as shown by MultiLogger, which logs the message to both loggers.

## Events

Events in C# are useful in the pub-sub (publisher and subscriber) design pattern. Events are a very powerful design pattern for asynchronous programming. An object can publish a set of events that subscribers in turn subscribe to. When the publisher raises the event, all subscribers are notified. The publisher raises the event without knowing who is listening to the events. Events are built using delegates, as shown below.

```
using System;
namespace DelegateExample
{
    public class ConsoleLogger
    {
        public  void WriteString(string s)
        {
            Console.WriteLine("Writing to console log: {0}", s);
        }
    }
    public class FileLogger
    {
        public  void LogString(string s)
        {
            Console.WriteLine("Logging to file log: {0}", s);
        }
    }
    public class DelegatesTest
    {
        public delegate void LogEventHandler(string s);              //
definition of the delegate.
        public static event LogEventHandler logEvent;                //
the signature of the event.
        public static void Main()
        {
            ConsoleLogger cl = new ConsoleLogger();                   //
create the first subscriber
            FileLogger fl = new FileLogger();                         //
the second subscribe

            logEvent += new LogEventHandler(cl.WriteString);          //
subscribe the event and hook up the
            logEvent += new LogEventHandler(fl.LogString);            //
event handlers
            logEvent("A new event");                                  //
raise event which will invoke handlers
```

```
            Console.ReadLine();
        }
    }
}
```

## Summary

C# is a strongly typed, object oriented programming language that uses static binding. Many Objective-C concepts map closely to corresponding C# concepts. This chapter is a quick introduction to how C# concepts map to those of Objective-C. It provides a starting point for Objective-C users and points to additional resources. Knowledge of object oriented programming, and Objective-C experience, will provide a strong foundation to enable you to master C# quickly.

# Chapter 5: Image Format Considerations in migration of iPhone applications to Windows Phone 7


Images play a critical role in today's mobile applications. Applications engage users visually with images rather than with use of the written word. It is important to account for resources such as images, video, and audio when you plan your Windows Phone 7 project.

If you are planning to migrate or rewrite your iPhone application to Windows Phone 7, it is important for you to reuse your existing images. However, due to differences in the format, size and resolution of images on the two platforms, you will not be able to use images from your iPhone application in your Windows Phone 7 application. You should consider redesigning the images or converting them. Redesigning images can be expensive whereas conversion of images can be time consuming.

In this blog, we will cover what developers and designers should know about the images in their Windows Phone 7 application. We will introduce the image sizes and formats required by the Windows Phone 7 and show how they compare with the iPhone image formats. This will help you plan your image conversion. We will provide some quick command line examples on how the ImageMagick tool can be used for the conversion. We provide an introduction to a Microsoft Image Converter tool, built on top of ImageMagick, that can be used to migrate your existing iPhone images to work in the Windows Phone environment.

## Critical role of Iconography in Windows Phone 7

Windows Phone 7 uses a unique design experience, called Metro, in the Windows Phone 7 OS and applications. The WP7 application design and User interface guidelines blog provides guidance on the look and feel of the application, including images and fonts. You are advised to pay close attention to images being designed for the Windows Phone Application. It is important that your application behaves consistently with the overall Windows Phone experience and stands out.

### Metro Image Guidelines

Metro UI design uses a clean, light and open interface design model which puts emphasis on content and not on the chrome. It uses flat, monochrome images for standard application tiles and toolbar icons. It recommends that developers use Metro design guidelines in their applications to present a consistent experience. You will need to redesign or convert some of your iPhone application images to be consistent with the Windows Phone experience.



Windows Phone

## Device resolutions

iPhone and Windows Phone devices use different screen sizes and screen resolutions. This can affect how your images will look on the phone. If you use an existing image out of your iPhone application and use it without modification in your Windows Phone 7 application, the result may not be pleasing.

| | Resolution | Size |
|---|---|---|
| iPhone | 480x320 | 3.5 inch diagonal |
| iPhone 4 | 960x640 | 3.5 inch diagonal |
| Windows Phone 7 | 800x480 | Varies from device to device |

As you can see from the above table, because of the different screen resolution, images designed for one platform cannot be used as is. They need to be - at least- resized to respect the aspect ratio of the target platform. The problem is even more complex if the images don't meet the resolution for the Windows Phone 7 Platform. The image conversion process is more complex to preserve an acceptable image quality.

## Differences in iPhone and WP7 Image Resolutions

Each platform has its own unique requirements when it comes to image sizes and their look and feel. As you migrate or rewrite your application for Windows Phone 7 platform, you will realize that you might be able to use some images without modification but others will need to be changed to make them look right on the new platform.

### Standard Image Requirements

The following table shows the comparison and association between iPhone and Windows Phone 7 image resolutions and types:

| iPhone image type | Resolution |
|---|---|
| Application icon | 57x57 or 114x114 for high resolution |
| N/A | |
| App store icon | 512x512 |
| Small icons for search and settings | 29x29 or 58x58 for high resolution |
| Tab bar icons | 30x30 |
| Tool bar / navigation bar | 20x20 |
| Launch image | 320x480 or 640x960 for high resolution |
| N/A | |

| Resolution | Windows Phone image type |
|---|---|
| 173x173 for tile | Application icon for tiles |
| 62x62 in app list | Application icon in app list |
| 99x99 small mobile artwork | Market place icons |
| 173x173 large mobile artwork | |
| 200x200 in PC marketplace catalog | |
| | N/A |
| 48x48 | Application bar icons |
| | N/A |
| 480x800 | Splash screen image or page background |
| 1000x800 | Background Art for Panorama control |

As you migrate your application to Windows Phone 7, you will need to scale the images up or down depending on the type and resolution of your images. Here are some samples of iPhone and Windows Phone 7 images.

### In App Images

There are two types of iPhone images – those for iPhone 3 or before and those for iPhone 4, which are also known as retina images. Retina images are higher resolution images. As we saw earlier, Windows Phone 7 screen resolution of 800x480 is higher than iPhone 3 resolution but lower than iPhone 4 resolution. Application images designed for iPhone will need to be upscaled to be used in Windows

Phone 7 applications. On the other hand, application images designed for the retina display will need to be downscaled.

### Windows Phone 7 Application Bar Images

Application bar icons on Windows Phone 7 should be 48 pixels by 48 pixels and must have a white foreground on a transparent background using an alpha channel. The Application Bar colorizes the icon according to the current style settings. Thus application bar icons must be designed appropriately for them to display correctly on Windows Phone 7.

### Other issues

Windows Phone 7 applications built using Silverlight handle images in JPEG or PNG format. GIF images are not supported in Silverlight. If your application uses GIF images, they need to be converted to either JPEG or PNG.

If your iPhone application PNG images are loaded via XCode, you should be aware that XCode optimizes these images while bundling them. Using or converting PNG files from your iPhone application bundle won't work properly. Use images from your XCode source directory and not the application bundle.

## Managing Images for Windows Phone 7 Projects

Let us look at different types of images required by Windows Phone 7 and understand what you need to create such images using your iPhone image resources

In this section we look at various Windows Phone 7 images and how to generate these images from your existing assets. One option is to use an open source tool called ImageMagick which is a very capable tool but can be difficult to use. We have given command line options for ImageMagic for each Windows Phone 7 images.

### Microsoft Image Converter Tool

Microsoft has released an image converter tool, built on top of ImageMagick, to make the process of converting images easier. It provides options to create various images required for Windows Phone 7 application and marketplace.

Unless specified, Windows Phone 7 images must be in PNG format. In the commands below, the target parameter should be a PNG file.

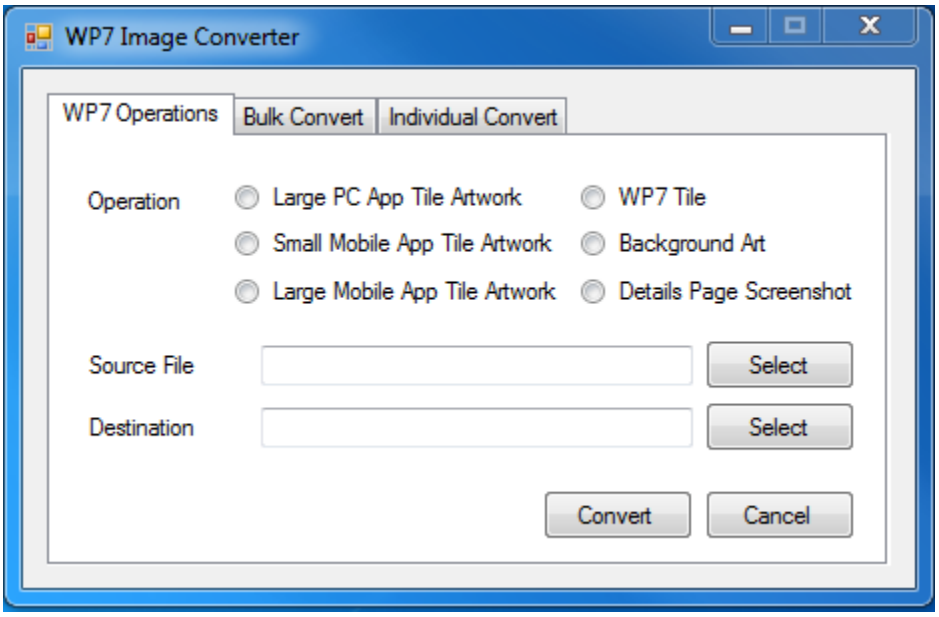
## Large PC App Tile Artwork

This 200x200 image is displayed in the Windows Phone 7 marketplace in the Zune PC client software. To convert an existing iPhone or other source image to this format, use the following options for the ImageMagick tool

Convert <source> -resize 200x200! -unsharp 0x1.1+.5+0  <target>

With resize geometry parameter of 200x200! ImageMagick converts the image to 200 x 200 and ignore the aspect ratio.

The same effect can be achieved using “Large PC App Tile Appwork” option in the Microsoft Image Converter tool.




## Small Mobile App Tile artwork

This image is used in the phone marketplace catalog. This image of size 99x99 is the smaller of the two images used in the phone marketplace catalog. This image can be created using the following ImageMagick convert options.

Convert <source>-resize 99x99 -unsharp 0x1.1+.5+0 <target>

You can also use Small Mobile App Tile Artwork option in the Microsoft Image Converter tool.



Windows Phone

## Large Mobile App Tile artwork:

This image is used in the phone marketplace catalog. This is the larger of the two images used in the phone marketplace catalog.

The same size image is also used in the **start experience** on the phone. An Image of size 173x173 is used when the user pins the application to Start on the phone. The image must be in the PNG format

To create convert the image using ImageMagick convert tool, use:

```
Convert <source> -resize 173x173 -unsharp 0x1.1+.5+0 <target>
```

You can use the “Large Mobile App Tile Artwork” option in the Microsoft Image Converter tool.

## WP7 Tile:

This image is used on the phone applist and must be a PNG image of size 62x62. To create convert the image using ImageMagick convert tool, use:

Convert <source> -resize 62x62 -unsharp 0x1.1+.5+0 <target>

You can also use the “WP7 Tile” option in the Microsoft Image Converter tool.

## Panorama Background

This panorama artwork becomes the background for your application if it is a featured app on the Marketplace. This is an optional image of size 1000x800. Use the following options to convert the image to this format:

Convert <source> -resize 1000x800! -unsharp 0x1.1+.5+0 <target>

You can also use the “Background Art“option in the Microsoft Image Converter tool.

## Details Page Screenshot

These images provide a preview of your app or game to users who are browsing your App details page on Windows Phone Marketplace. You must provide minimum one or maximum of 8 such images. Use the following options to convert the image to this format:

Convert <source> -resize 480x800! -unsharp 0x1.1+.5+0  <target>

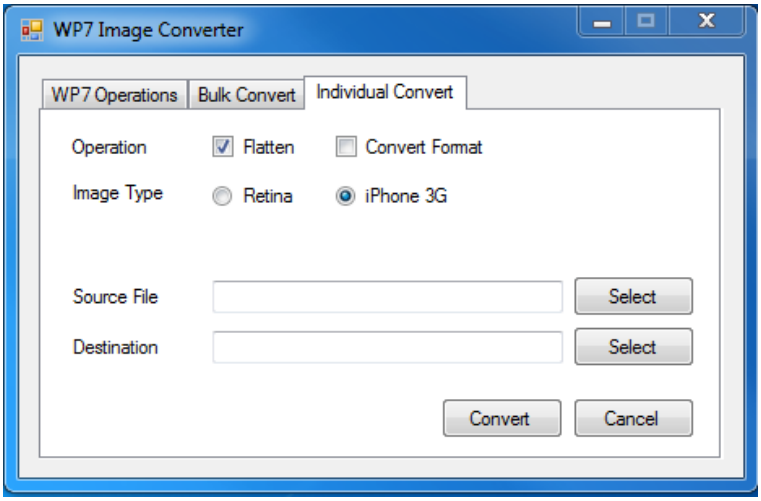
You can also use the “Details Page Screenshot“option in the Microsoft Image Converter tool.

## Flatten Images for application bar icons

Windows Phone 7 application bar icons are flat monochrome images with transparent background and white foreground, as described above. You can use the following ImageMagick options to create such images from iPhone tab bar icons which are monochrome. If you are transforming other multi-chrome images, you will need to use different options.

convert.exe <source> ( +clone )  -compose Difference -composite -modulate 100,0 +matte -threshold 0 -fill black +opaque rgb(255,255,255) -transparent black  <target>

Or you can use the “flatten” option in the Microsoft Image Converter . If you are converting from an iPhone 4 image, use Retina option otherwise choose iPhone 3G option.




## Convert Image Format

Since Windows Phone 7 does not support GIF images, you will need to convert your GIF images to PNG (or JPEG). Use the following option for converting your image format.

Convert <source> <target>

You may use the “Convert Format” to convert the format of images using Microsoft Image Converter tool.

## Resize Images

Due to differences in the iPhone and Windows Phone 7 screen size, screen resolution and aspect ratio, you will need to convert your other image assets. Use the following ImageMagick convert options to resize the images

Convert <source> -resize <app geometry>  <target>

You may use the Microsoft Image Converter tool to resize images.  Use Retina if you are resizing the image from iPhone 4 project or iPhone 3G otherwise.

### Learn Further about the Windows Phone 7 Image Format

UI Design and Interaction Guide for Windows Phone 7 and Windows Phone 7 Certification Guide provide definitive requirements for the resolution and types of images required for each Windows Phone 7 applications. The UI design guide provides guidance on the design of application bar icons.

### Design New Images

Some of the WP7 images such as application tile or application list are particularly important since they always remain visible to users. They must be high quality and engaging. You may decide to redesign these images or start from the converted images and modify them further to suit your needs. Windows Phone 7 images may be designed using Microsoft Expression Design or Adobe Photoshop.

## Conclusion

ISVs who want to migrate or rewrite their applications for Windows Phone 7 need to be aware that their images may need to be transformed to make them captivating on the Windows Phone. The differences in images on two platform are due differences in the screen resolutions and as well as in the standard image formats chosen for each platform. Redesigning the images may be an expensive and time consuming option. You can start by converting existing images using open source tool such as ImageMagick and customize them further.

## Resources

1. Windows Phone 7 UI Design and Interaction Guide
2. Windows Phone 7 Application Certification Requirements
3. Microsoft's Windows Phone 7 Interoperability site
4. Microsoft iPhone to Windows Phone 7 Image Conversion Tool
5. ImageMagick tool

# Chapter 6: Application Lifecycle Differences Between Windows Phone 7 and the iPhone

In this chapter, we are going to look at the navigation model of the Windows Phone 7. We will examine the various application states needed to support the navigation model, and what the developer needs to do to support those application states and the transitions between them. We will then look at what the developer needs to do to duplicate iPhone multitasking.

## iPhone and Windows Phone 7 Navigation Models

Technically speaking, both the iPhone and the Windows Phone 7 allow only **one** application to execute *in the foreground* at a time. The foreground application owns the screen and receives the touch events. Let us briefly review the iPhone execution model.

### iPhone Execution Model

With iOS4, the iPhone introduced multitasking, with the ability to do fast application switching. When the user clicks the home button, instead of terminating, the currently executing application is put into a background running state. In this state, the application is expected to save the data. If the user switches back to this application, after using another application, the background application transitions to foreground and is made active again, at which time the user can resume where he/she was. Additionally, iOS4 also supports other forms of multitasking, such as task completion or background execution for applications using audio, location or VOIP.

### Windows Phone 7 Navigation Model

In Windows Phone 7, the user can switch to another application in two different ways. The user can hit the start button at any time to reach the Start page and launch another application. The user can also use the back button to navigate out of an application.

For example, consider that the user is composing a blog post, using the WordPress application, and then hits the start button to reach the Start screen, at which time the WordPress application is deactivated. The user then opens up the Facebook application. At this time, the Facebook application is launched.

The hardware back button allows the user to navigate back between pages within an application, or across applications. In the above example, the user can hit the back button while in the Facebook application, to first reach the WP7 Start screen, and hit the back button again to go back to the WordPress application. The WordPress application would open with the compose screen, exactly how and where the user left it.

While WP7 does not support actual multitasking, the WP7 navigation model allows a natural navigation

much like the browser back button. The application state is preserved as the user navigates across applications using the back button.

The following table provides an overview of various events and the behavior of the application on iOS and Windows Phone 7.

| User Action or Event | iOS 4.0 Behavior | WP7 Behavior |
|---|---|---|
| An incoming phone call or SMS | Running application is moved to the background but running | Running application is deactivated |
| User Clicks the home button | Running application is moved to the background but running | Running application is deactivated |
| User Clicks another application from the multitasking menu | Background application is moved to foreground and made active to its original state | - |
| Navigation between applications using back button | - | Deactivated or tombstoned application is activated to its original state |

## Programming for application States and navigation

On both the iPhone and WP7, the developers need to take certain steps to support the application life cycle.

### iPhone support for multitasking

In order to support fast application switching, iPhone application developers have certain responsibilities. When the application is moved to the background, they need to save the application state, which is then restored when the application is subsequently moved again to the foreground.

In particular, the application moving to background will receive a callback, `applicationDidEnterBackground`, at which time the application should do the bookkeeping, save the state and reduce the application memory footprint. When the user relaunches the application using the multitasking UI, the application will receive the `applicationWillEnterForeground` callback at which time the application should restore the state.

When the application is launched from the application icon, the application receives the `applicationDidBecomeActive` callback, at which time the application can initialize the state. When the application becomes inactive, due to an interrupt such as a phone call, it receives the `applicationWillResignActive` event at which time, so it can save the state, as it may be moved to the background.

## Windows Phone 7 LifeCycle

**Launching the Application** When the user launches the application for the first time, the application receives the `Application_Launching` event. In order to provide fast startup response, the application should do little work in this event handler. In particular, it should avoid any web downloads or isolatedStorage (see below) data fetch operations. Once active, it can initialize the state or load any saved state.

**Terminating the Application** While the application is running, the user may terminate it by navigating out of the application using the back button. At this time, the application will receive the `Application_Closing` event. In response, the application should perform any cleanup and save the persistent application data to isolatedStorage.

**Deactivating the Application and Tombstoning** While the application is running, the user can hit the Start button or launch another application via launchers or choosers. The user may launch the browser by clicking on a link in the application.

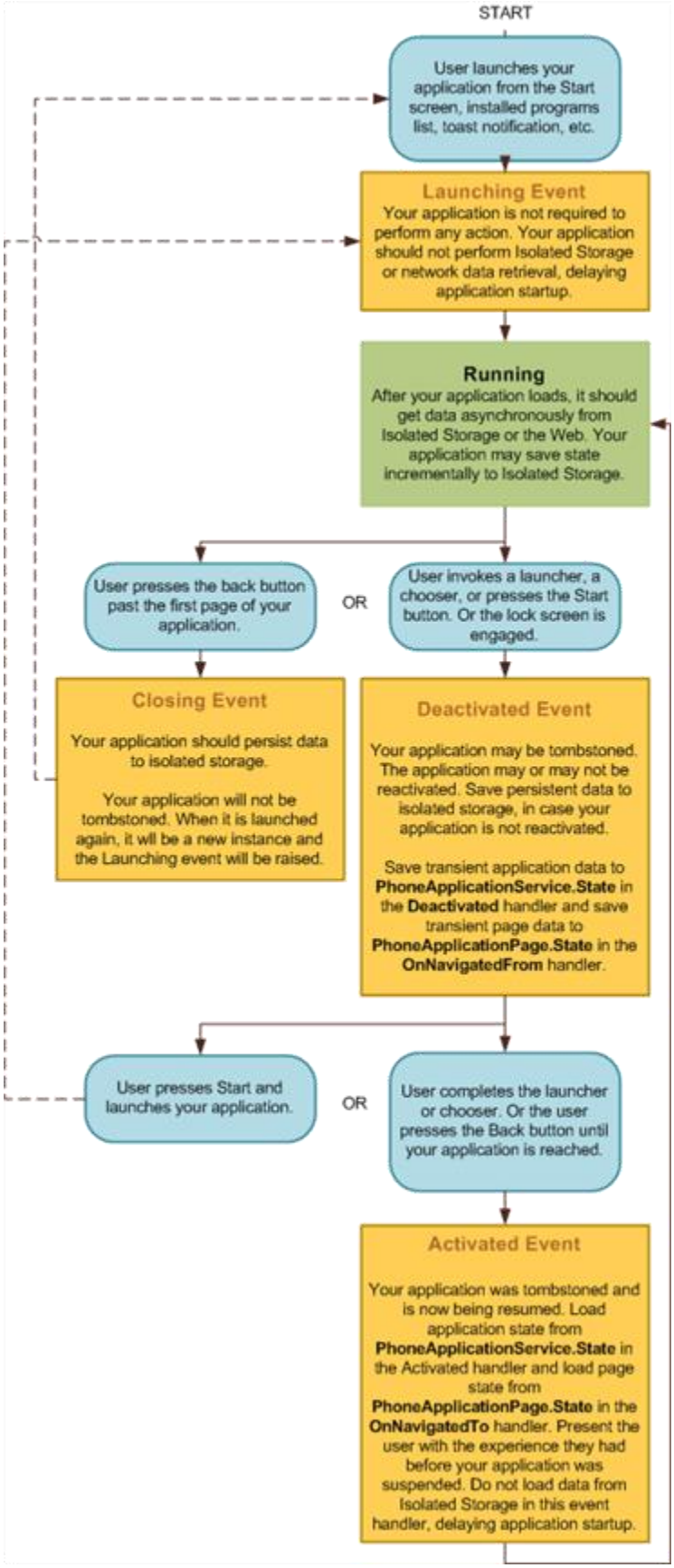
Similar to an application that is closed, an application that is deactivated is also terminated. However, unlike a closed application, for a deactivated application, the OS stores a record (a tombstone) for the state of the application. This is maintained as part of the application back stack which is used to facilitate navigation using the back button.

In these cases, the application is sent an `Application_Deactivated` event, at which time the application should save all persistent data to isolatedStorage and all transient data, such as the values of page fields, using PhoneApplicationPage.state.

**Reactivating the Application** Upon completing the launcher, or the user navigating back into an application using the back button, the application will be reactivated. Upon reactivation, the application will receive the `Application_Activated` event. Since the application is being reactivated from a tombstone state, the application should load the persistent state data from the isolatedStorage and the transient state data from PhoneApplicationPage.state.

### Windows Phone 7 Application State Transition Diagram

The following state diagram shows the various states and explains what the developer should do in response to various events.


START
User launches your application from the Start screen, installed programs list, toast notification, etc.
Launching Event
Your application is not required to perform any action. Your application should not perform Isolated Storage or network data retrieval, delaying application startup.
Running
After your application loads, it should get data asynchronously from Isolated Storage or the Web. Your application may save state incrementally to Isolated Storage.
User presses the back button past the first page of your application.
OR
User invokes a launcher, a chooser, or presses the Start button. Or the lock screen is engaged.
Closing Event
Your application should persist data to isolated storage.
Your application will not be tombstoned. When it is launched again, it will be a new instance and the Launching event will be raised.
Deactivated Event
Your application may be tombstoned. The application may or may not be reactivated. Save persistent data to isolated storage, in case your application is not reactivated.
Save transient application data to PhoneApplicationService.State in the Deactivated handler and save transient page data to PhoneApplicationPage.State in the OnNavigatedFrom handler.
User presses Start and launches your application.
OR
User completes the launcher or chooser. Or the user presses the Back button until your application is reached.
Activated Event
Your application was tombstoned and is now being resumed. Load application state from PhoneApplicationService.State in the Activated handler and load page state from PhoneApplicationPage.State in the OnNavigatedTo handler. Present the user with the experience they had before your application was suspended. Do not load data from Isolated Storage in this event handler, delaying application startup.

# WP7 LifeCycle and Tombstoning Example

Let us look at a simple example that illustrates the Windows Phone 7 state transitions and tombstoning. This is a one page shopping list application where the user can add items to the list. When the user leaves the application, the shopping list is saved automatically.

## Saving the Application State

On the iPhone, you may be saving the state in a number of different ways. On the iPhone, using NSUserDefaults, using files in the Documents folder of the application, or using SQLLite are all possible ways to save application state data.

To save the persistent state of the application on WP7, i.e., the shopping list, we are going to use isolatedStorage. isolatedStorage is a safe storage that is accessible only to that application. It ensures that one application cannot affect another application. To save the state, we first get the isolatedStore for the application. We then create a file in isolatedStorage, in which to save the persistent state. To save the shopping list, we first serialize the list and then save it to the file.

```
public static void SaveShoppingList(ShoppingListInfo shoppingListInfo, string fileName)
{
     //Get the isolatedStore for this application
     using (IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication())
     {
         // create a new file
         using (IsolatedStorageFileStream fs = isf.CreateFile(fileName))
         {
              //and serialize data and save it
              XmlSerializer xmlSer = new XmlSerializer(typeof(ShoppingListInfo));
              xmlSer.Serialize(fs, shoppingListInfo);
         }
     }
}
```

In order to save transient state, WP7 provides another class called PhoneApplicationService.State. We will see the use of this object below.

## Terminating the application

When the user uses the back button to leave the application, the application is terminated. What the application should save during termination depends upon the nature of the application. In this example, we will save the work without asking the user so that when the user next opens the application, the shopping list is intact. In this example, we will not save any item that the user was typing in the item textbox. We use the helper method defined earlier to save the shopping list. Such cleanup and state saving may be performed in response to the Application_Closing event instead of OnBackKeyPress.

```
protected override void OnBackKeyPress(System.ComponentModel.CancelEventArgs e)
{
     base.OnBackKeyPress(e);
     //do not save what is in the item, i.e. it is transient
     txtItem.Text = "";
     //preserve data in persistent store
     Utils.SaveShoppingList((App.Current.RootVisual as PhoneApplicationFrame).DataContext as
ShoppingListInfo,
               "ShoppingListInfo.dat");
     }
}
```

This event is comparable to the ApplicationWillTerminate callback in versions prior to iOS4. In iOS4, there is no equivalent to this operation, as the state of the application will already be saved in response to the applicationDidEnterBackground callback.

## Application Launching

When the application is launched from the Start screen, the application received the Application_Launching event. This is equivalent to the application receiving `applicationDidFinishLaunchingWithOptions` and `applicationDidBecomeActive` callbacks in iOS4.

During launch, we will examine whether any persistent data is available. If we find the persistent data, we will load it into the txtList textbox to preserve the shopping list.  We first get the isolatedStore for the application. Using the isolated store, we check if the ShoppingListInfo.dat file exists. If it is available, we deserialize the data that was stored in the file and reload the ShoppingListInfo. The fields in the application are databound using the last line in this snippet. If you run the application, you will find that the shopping list is preserved, upon relaunching the application.

```
        // Code to execute when the application is launching (eg, from Start)
        // This code will not execute when the application is reactivated
        private void Application_Launching(object sender, LaunchingEventArgs e)
        {
            //Trace the event for debug purposes
            Utils.Trace("Application Launching");
            //Create new data object variable
            ShoppingListInfo shoppingListInfo = null;
            //Try to load previously saved data from IsolatedStorage
            using (IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication())
            {
                //Check if file exits
                if (isf.FileExists("ShoppingListInfo.dat"))
                {
                    using (IsolatedStorageFileStream fs = isf.OpenFile("ShoppingListInfo.dat",
System.IO.FileMode.Open))
                    {
                        //Read the file contents and try to deserialize it back to data object
                        XmlSerializer ser = new XmlSerializer(typeof(ShoppingListInfo));
                        object obj = ser.Deserialize(fs);

                        //If successfully deserialized, initialize data object variable with it
                        if (null != obj && obj is ShoppingListInfo)
                            shoppingListInfo = obj as ShoppingListInfo;
                        else
                            shoppingListInfo = new ShoppingListInfo();
                    }
```

```
                }
                else
                    //If previous data not found, create new instance
                    shoppingListInfo = new ShoppingListInfo();
            }

            //Set data variable (either recovered or new) as a DataContext for all the pages of
the application
            RootFrame.DataContext = shoppingListInfo;
        }
```


## Application Deactivation

Let us examine what happens when the user hits the start button while running the application. When the user hits the home button in the iPhone, the application receives the applicationDidEnterBackground callback. On WP7, the application will receive the `Application_Deactivated` event. In response to this event, we will save the entire application state, i.e. the shopping list as well as the anything entered in the item textbox. If this application is reactivated, we will restore both textboxes to provide the same experience as if the user had navigated back into the application.

We will also save the transient application state; for this, we will use the PhoneApplicationService.State object.

```
        // Code to execute when the application is deactivated (sent to background)
        // This code will not execute when the application is closing
        private void Application_Deactivated(object sender, DeactivatedEventArgs e)
        {
            //Trace the event for debug purposes
```

```
            Utils.Trace("Application Deactivated");

            //Add current data object to Application state
            PhoneApplicationService.Current.State.Add("UnsavedShoppingListInfo",
RootFrame.DataContext as ShoppingListInfo);
        }
```

## Application Activation

When the user uses back button to navigate into the application, the application receives the `Application_Activated` event. In response to this event, we will reload the data from the `PhoneApplicationService.State` object. If we find the necessary data, we will load the UI elements using that data. If we do not find any saved data, the fields will get initialized to blanks. If you run the application, then hit the home key and navigate back into the application using the back button, you will find that both the shopping item and the shopping list are preserved.

```
        // Code to execute when the application is activated (brought to foreground)
        // This code will not execute when the application is first launched
        private void Application_Activated(object sender, ActivatedEventArgs e)
        {
            //Trace the event for debug purposes
            Utils.Trace("Application Activated");

            //Create new data object variable
            ShoppingListInfo shoppingListInfo = null;

            //Try to locate previous data in transient state of the application
            if (PhoneApplicationService.Current.State.ContainsKey("UnsavedShoppingListInfo"))
            {
                //If found, initialize the data variable and remove in from application's state
                shoppingListInfo =
PhoneApplicationService.Current.State["UnsavedShoppingListInfo"] as ShoppingListInfo;

                PhoneApplicationService.Current.State.Remove("UnsavedShoppingListInfo");
            }

            //If found set it as a DataContext for all the pages of the application
            //An application is not guaranteed to be activated after it has been tombstoned,
            //thus if not found create new data object
            if (null != shoppingListInfo)
                RootFrame.DataContext = shoppingListInfo;
            else
                RootFrame.DataContext = new ShoppingListInfo();
        }
```


## iOS and Windows Phone 7 State and Event mapping

The following table provides mapping between iOS callbacks and Windows Phone 7 events related to the application lifecycle. While Windows Phone 7 does not provide multitasking, support for navigation using the back button requires that the application save the state during application deactivation and reload it when the user reactivates the application.

| iOS Callback | Windows Phone 7 Event | Notes |
|---|---|---|
| `applicationDidFinishLaunchingWighOptions` | - | |
| `applicationDidBecomeActive` | `Application_Launching` | Load Application persistent state |
| `applicationWillResignActive` | - | |
| `applicationDidEnterBackground` | `Application_Deactivated` | Save application state |
| `applicationWillEnterForeground` | `Application_Activated` | Reload application state |
| `applicationWillterminate` | `Application_Closing` | Save Application Persistent state |

## Summary



Windows Phone

In this chapter we looked at the Windows Phone 7 application states and transitions between them. We looked at what the developer needs to do to support the Windows Phone 7 navigation model. While the application model differs between the iPhone and Windows Phone 7, the developer needs to preserve the application state in much the same manner when the current application is moved from the foreground and is deactivated.

## Resources

1. Introducing the Windows Phone Application Life Cycle—Tombstoning
2. Execution Model for Windows Phone
3. Windows Phone Application Lifecycle

# Chapter 7: iPhone to Windows Phone 7 Application Preference Migration

## Application Preferences

Preferences are application-specific settings that define the look and feel and behavior of an application. The user can update the preferences at any time to change the behavior of the application.

Both iPhone and Windows Phone provide easy means to update application settings. They both follow a similar philosophy of ease of use and simplicity. However, they follow different ways for presenting and implementing application preferences.

## iPhone Application Preferences

On the iPhone, applications have two options for presenting preferences.

- Display preferences inside the application.
- Manage preferences from the system wide Settings application.

While the developer can use either mechanism to implement the preferences, the second way to manage application preferences via the Settings application is the preferred one. If the developer expects that the settings will be updated frequently, it may be better to include the preferences within the application. Developers may use both mechanisms.

The Settings application implements a hierarchical set of pages for managing application preferences. The main page shows the system preferences and links to the pages for third party applications. Selecting the application preferences link takes the user to the preferences page for that application. We will look at how to migrate the iPhone application preferences presented using the preferred way, i.e. via the Settings application.

## Windows Phone 7 Application Preferences

Windows Phone 7 provides a Settings application for system and system application settings. In contrast to iPhone, third party application settings cannot be placed within this system Settings application. Application settings must be part of the application. This provides consistency and ease of use; users are not confused about where to look for the settings and avoids having to look for settings in both central settings application and the application pages.

Windows Phone 7 UI Design and Interaction Guide recommends that developers keep the application settings brief and clear to improve the usability, and also that they avoid complex multi-level, multi-page application settings. As such, the application should be designed for the most common use. If the application has several user-selectable settings, it should create a separate settings page within the application. This allows the user to update the settings without leaving the application.

Changes to application settings should be applied immediately without confirmation. This means that confirmation dialog such as “OK” is not needed. However, you may inform the user that the settings have been updated. Any settings that that cannot be undone or overwrite or delete data should allow the user to opt out by canceling the change.

Comparison between the two platforms

The following table provides the comparison between the two platforms.

| Purpose | iPhone | Windows Phone 7 |
|---|---|---|
| **Application Preferences** | System wide settings application<br>In-Application preference pages | In Application preference page |
| **Preferred/Required** | System wide settings application | In-app preferences required |
| **Preference UI** | Declarative syntax for System wide Settings<br><br>Developer implements a page for in-app preferences | Developer implements in-app page for in-app preferences |
| **Storage mechanism** | Application specific file store | IsolatedStorage |
| **Preferences** | Saved as key-value pairs | Saved as key- value pairs |

# Migrating Application Preferences

## Overview of iPhone Application Preferences

iPhone provides a Settings bundle to manage preferences from the Settings application. The Settings bundle consists of Root.plist and other .plist files as well as files that store localized string resources. The system uses the .plist files to present the UI for the application preference page. The Settings page .plist file consists of specifications for preference parameter types. For the preferences included within the application, the developer needs to implement the view to access and manipulate the preferences.

Regardless of how the application preferences are presented to the user, the application developer uses NSUserDefaults class to access the preferences from the code. The NSUserDefaults class provides a dictionary to store key-value pairs which are persisted in the .plist file in the application-specific file store.

## Windows Phone 7 Application Preferences

### *Application Preference UI*

In Windows Phone 7 the developer needs to implement the pages for accessing and manipulating application preferences. The settings pages are no different from any other application page and one can use all available Windows Phone UI widgets to manipulate the preferences.



Let us look at the types of iPhone application settings and how they can be migrated to Windows Phone 7. The following table shows iPhone preference types and the types of controls. The right hand column shows the corresponding Windows Phone 7 controls that can be used to migrate them.

| iPhone Preference control | iPhone preference type | Purpose | Corresponding Window Phone Control type |
|---|---|---|---|
| Text Field | PSTextFieldSpecifier | Editable text field for string parameter | TextBox |
| Title | PSTitleValueSpecifier | Read-only string value | TextBlock |
| Toggle switch | PSToggleSwitchSpecifier | Preference that can have only two values | Checkbox |
| Slider | PSSliderSpecifier | Preference that represents range of values | Slider |
| Multi-value | PSMultiValueSpecifier | Selection of one value from a list | RadioButtons or ListBox |
| Group | PSGroupSpecifier | Organize collection of preferences together | StackPanel, Grid, or Table |
| ChildPane | PSChildPaneSpecifier | Navigate to a different page of preferences | Button, link to navigate |

### *Persisting User Preferences*

Windows Phone 7 uses isolatedStorageSettings class. IsolatedStorageSettings is implemented using IsolatedStorage which provides complete isolation between applications. IsolatedStorage provides safety as one application cannot access settings of other applications or affect other applications. IsolatedStorageSettings class provides a dictionary to store preferences as key value pairs.

Application preferences presented via the preference page can be written to IsolatedStorage. The following code snippet shows how to write the settings to IsolatedStorage. All application preferences are stored as key-value pairs where the key represents the name of the preference.

```
// isolated storage settings
IsolatedStorageSettings isolatedStoreSettings;
try
{
    // Get the settings for this application.
    isolatedStoreSettings = IsolatedStorageSettings.ApplicationSettings;
}
catch (Exception e)
{
    Debug.WriteLine("Exception getting IsolatedStorageSettings.ApplicationSettings: " + e.ToString());
}

// Save the application settings
try
{
    // check if the key exists. If not create it
    if (!isolatedStoreSettings.Contains(Key))
    {
        isolatedStoreSettings.Add(Key, value);
    }
    else
    {
```

```
        // if new value is different, set the new value.
        if (isolatedStoreSettings[Key] != value)
        {
            isolatedStoreSettings[Key] = value;
        }
    }
}
catch (Exception e)
{
    Debug.WriteLine("Exception in IsolatedStoreSettings: " + e.ToString());
    throw e;
}
```

On the iPhone, the Settings application handles the persisting of application settings to the application defaults database. The defaults database is provided via the on-device .plist file. However, if your application presents preferences via application pages, you will see that the Windows Phone 7 isolated storage is similar to NSUserDefaults.

```
// get the handle to application NSUserDefaults
NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];
// saving a string preference value to a key named “nickname”
[prefs setObject:@"DarthVader" forKey:@"nickname"];
// synchronize to persist them to file store
[prefs synchronize];
```

### *Reading Application Preferences*

iPhone uses NSUserDefaults to access application preferences. Here is a typical code snippet to access a preference.

```
// get the handle to application NSUserDefaults
NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

// getting an NSString preference value
NSString *UserNickname = (NSString *)[prefs objectForKey:@"nickname"];
```

The corresponding code snippet for retrieving application settings is shown below. As you can see, the iPhone application retrieval code can be migrated easily to Windows Phone 7.

```
string UserNickname;
try
{
    UserNickName = (string)isolatedStoreSettings["nickname"];
}
catch (KeyNotFoundException)
{
    Debug.WriteLine("Nickname not found");
}
catch (Exception e)
{
    Debug.WriteLine("Exception in reading from IsolatedStorageSettings: " + e.ToString());
}
```

| Purpose | iPhone mechanism | Windows Phone 7 mechanism |
|---|---|---|
| Preference store | NSUserDefaults backed by application file store | Persistent application specific store: Isolated Storage |
| Access persistent store | `[NSUserDefaults standardUserDefaults]` | `IsolatedStorageSettings.ApplicationSettings;` |
| Write Preference | `[[NSUserDefaults standardUserDefaults] setObject:value forKey:@"Key"];` | `isolatedStoreSettings.Add("Key", value);`<br><br>`isolatedStoreSettings["Key"] = value;` |
| Read Preference | `// getting a preference object`<br>`[prefs objectForKey:@"Key"];` | `isolatedStoreSettings["Key"];` |

## Migration Sample

Let us look at migrating simple application preferences from iPhone to Windows Phone 7. In our ReadPoetry application, there are only two preferences, namely, font size and night mode. In the normal mode, the application uses black fonts on a white background whereas in the night mode it uses white fonts on a black background.


### *Create Initial Application*

Open Visual Studio 2010 for Windows Phone and create a new application using Windows Phone Application template. Name it ReadingPoetry and click **OK**.

Open MainPage.xaml using Solution Explorer. Right click on the application title and select view XAML to edit the title to "READING POETRY". Similarly, edit the Page Title to "jabberwocky". Your TitlePanel XAML should look like following:

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
     <TextBlock x:Name="ApplicationTitle" Text="READING POETRY" Style="{StaticResource PhoneTextNormalStyle}"/>
     <TextBlock x:Name="PageTitle" Text="jabberwocky" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

In the ContentPanel Grid, add the following TextBlock with the poem in it (with all those LineBreaks). Your XAML should now look like this:

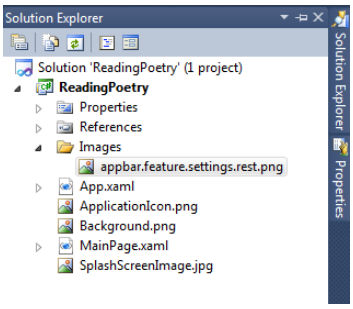
```
<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <TextBlock Name="PoetryBlock">
        `Twas brillig, and the slithy toves<LineBreak/>
        Did gyre and gimble in the wabe:<LineBreak/>
        All mimsy were the borogoves,<LineBreak/>
```

```
        And the mome raths outgrabe.<LineBreak/>
        <LineBreak/>
        "Beware the Jabberwock, my son!<LineBreak/>
        The jaws that bite, the claws that catch!<LineBreak/>
        Beware the Jubjub bird, and shun<LineBreak/>
        The frumious Bandersnatch!"<LineBreak/>
        <LineBreak/>
    </TextBlock>
</Grid>
```

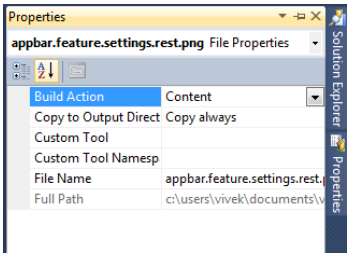
You can now run your application with ‘F5’ to see the application display first two stanzas of Jabberwocky.

### *Add Application Bar*

We will first add an icon for application settings to our application. In Solution Explorer, right click ReadingPoetry project and select **Add** followed by **New Folder** and rename it to **Images**. Using Windows Explorer, find your Windows Phone SDK Icons directory (typically at C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v7.0\Icons) and copy appbar.feature.settings.rest.png to newly created Images folder from the "dark" subdirectory.




Right click on appbar.feature.settings.rest.png in Solution Explorer and select **Properties**. Change the Build Action to "**Content**" and Copy to Output Directory to "**Copy Always.**"



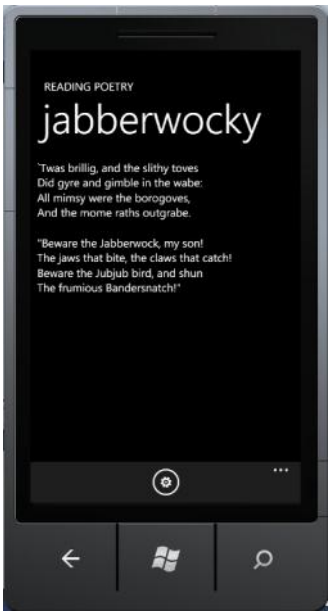

Now we will add an Application Bar to our application. Uncomment XAML underneath "<!--Sample code showing usage of ApplicationBar-->" to add the application bar. Delete the Second ApplicationBarIconButton as well as the entire ApplicationBar.MenuItems node. Update the IconUri for the first ApplicationBarIconButton to `"/Images/appbar.feature.settings.rest.png".`

```
Your XAML should look like following
<!--Sample code showing usage of ApplicationBar-->
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="False">
        <shell:ApplicationBarIconButton IconUri="/Images/appbar.feature.settings.rest.png"
Text="Settings"/>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```



Windows Phone

Hit F5 to run the application again and now it should show the poem along with the application bar at the bottom as shown below. Of course, if you click on the settings icon, nothing will happens which is expected.In the next section we will create an event handler for it.




## *Create Settings Page*

Stop the application and right click on the ReadingPoetry project in Solution Explorer and click **Add** followed by **New Item** and select **Windows Phone Portrait Page** and change its name to "Settings.xaml"**.** Settings.xaml should open up in Visual Studio.

Follow the above procedure to update the ApplicationTitle TextBlock to "SETTINGS" and PageTitle TextBlock to "reading poetry."

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="SETTINGS" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="reading poetry" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

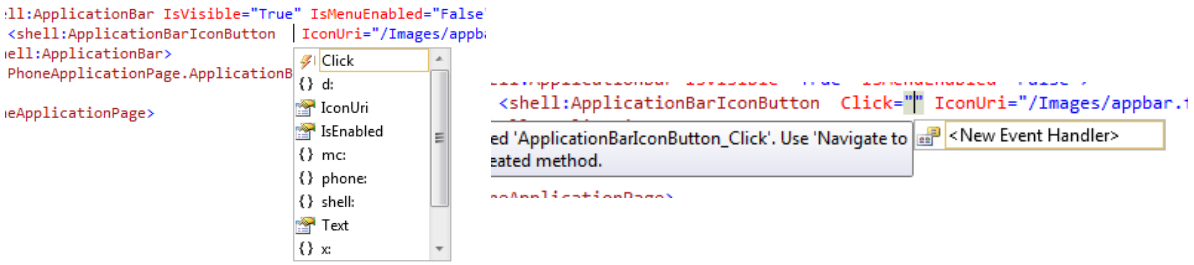
We will now migrate iPhone application preferences to our Windows Phone 7 application. We will use a check box instead of the toggle switch used for Night Mode. On the other hand, we will use radio buttons instead of multi-value control for font size.

Use TextBlock, CheckBox and RadioButtons to create the user interface for the settings page. The XAML should look like the following:

```
<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <TextBlock Text="Mode" FontSize="28"></TextBlock>
        <CheckBox FontSize="24" Margin="30,0,0,0" Name="NightModeCheckbox" Content="Night Mode" />
        <TextBlock Text="Font Size" FontSize="28"></TextBlock>
        <RadioButton Name="FontSizeSmallBtn" Margin="30,0,0,0" GroupName="FontSizeGroup" FontSize="24" Content="Small"/>
        <RadioButton Name="FontSizeMedBtn"  Margin="30,0,0,0" GroupName="FontSizeGroup" FontSize="24" Content="Medium"/>
        <RadioButton Name="FontSizeLargeBtn"  Margin="30,0,0,0" GroupName="FontSizeGroup" FontSize="24" Content="Large"/>
    </StackPanel>
</Grid>
```

Now we will hook up the new page with the settings icon.

Go back to MainPage.xaml and in the ApplicationBarIconButton tag and type space at the end and select **Click** and type **return** at which time Visual Studio will offer to create a New Event Handler.




Select **New Event Handler** and it will create ApplicationBarIconButton_click event handler in the code behind. Your XAML for ApplicationBarIconButton should look like this

```
<shell:ApplicationBarIconButton Click="ApplicationBarIconButton_Click"
IconUri="/Images/appbar.feature.settings.rest.png" Text="Settings"/>
```

Right click on XAMLand select **View Code** which will open MainPage.xaml.cs file which should include a method `ApplicationBarIconButton_Click`. Add the following code so that your method look like this:

```
private void ApplicationBarIconButton_Click(object sender, EventArgs e)
{
    this.NavigationService.Navigate(new Uri("/Settings.xaml", UriKind.Relative));
}
```

Hit F5 again to run the application. If you click on the settings icon, it will open the settings page as shown below. You can click on the settings but nothing will happen which is expected as we do not have code to save or retrieve settings.


### *Saving and Retrieving Application Settings*

Right click on ReadingPoetry project in Solution Explorer and select **Add** followed by **Class.** In the **add new item** dialog, type **AppSettings.cs** as the name of the class.

Declare IsolatedStorageSettings as shown below and initialize it in the class constructor as shown below.

```
// isolated storage settings
IsolatedStorageSettings isolatedStoreSettings;

// Constructor for the application settings.
public AppSettings()
{
    try
    {
        // Get the settings for this application.
        isolatedStoreSettings = IsolatedStorageSettings.ApplicationSettings;
    }
    catch (Exception e)
    {
        Debug.WriteLine("Exception getting IsolatedStorageSettings.ApplicationSettings: " + e.ToString());
    }
}
```

Now write three routines to write, read and save the application settings.

```
// Update the settings. If the setting does not exist, then add the setting.
public bool AddUpdateSetting(string Key, Object value)
{
    bool updated = false;
    try
```

```
    {
        if (!isolatedStoreSettings.Contains(Key))
        {
            isolatedStoreSettings.Add(Key, value);
            updated = true;
        }
        else
        {
            // if new value is different, set the new value.
            if (isolatedStoreSettings[Key] != value)
            {
                isolatedStoreSettings[Key] = value;
                updated = true;
            }
        }
    }
    catch (Exception e)
    {
        Debug.WriteLine("Exception in IsolatedStoreSettings: " + e.ToString());
        throw e;
    }
    return updated;
}
```

In the read routine, we will get the value from the isolateStoreSettings dictionary using our key

```
// Get the current value of the setting, or if not found, set  to the  setting.
public valueType GetSettingValue<valueType>(string Key, valueType val)
{
    try
    {
        if (!isolatedStoreSettings.Contains(Key))
        {
            isolatedStoreSettings.Add(Key, val);
            return val;
        }
        else
        {
            return (valueType) isolatedStoreSettings[Key];
        }
    }
    catch (Exception e)
    {
        Debug.WriteLine("Exception in IsolatedStoreSettings: " + e.ToString());
        throw e;
    }
}
```

Finally, the save routine will save the settings to the persistent store.

```
// Save the settings.
public void Save()
{
    isolatedStoreSettings.Save();
}
```

With these routines in place, we will write two properties to save and retrieve our NightModeSettings and FontSizeSettings as shown below.

```
// isolated storage key names of our settings
const string NightModeSettingKeyName = "NightModeSetting";
const string FontSizeSettingKeyName = "FontSizeSetting";

// default values for our settings
const bool NightModeSettingDefault = false;
const string FontSizeSettingDefault = "medium";

// Property to get and set a NightMode Setting.
public bool NightModeSetting
{
    get
    {
        return GetSettingValue<bool>(NightModeSettingKeyName, NightModeSettingDefault);
    }
    set
    {
        AddUpdateSetting(NightModeSettingKeyName, value);
        Save();
    }
}
// Property to get and set a RadioButton Setting Key.
public string FontSizeSetting
{
    get
    {
        return (GetSettingValue<string>(FontSizeSettingKeyName, FontSizeSettingDefault));
    }
    set
    {
        AddUpdateSetting(FontSizeSettingKeyName, value);
        Save();
    }
}
```

With this, we have the plumbing for retrieving and persisting preferences.

### *Saving the Settings*

Open Settings.xaml.cs file using Solution Explorer and edit the Settings class constructor to create appSettings instance.

```
AppSettings appSettings;
public Settings()
{
    InitializeComponent();
    // create AppSettings instance
    appSettings = new AppSettings();
}
```

When the user navigates away from this page, by clicking back button, OnNavigatedFrom event is fired. In this event handler, we will save the user selections to our application settings.

```
protected override void OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
{
    //save settings based on selections in the UI
    appSettings.NightModeSetting = ((bool)NightModeCheckbox.IsChecked);
    if ((bool) FontSizeSmallBtn.IsChecked) appSettings.FontSizeSetting = "small";
    if ((bool)FontSizeMedBtn.IsChecked) appSettings.FontSizeSetting = "medium";
    if ((bool)FontSizeLargeBtn.IsChecked) appSettings.FontSizeSetting = "large";
    base.OnNavigatedFrom(e);
}
```

Similarly when the user navigates to this page, by clicking settings icon, OnNavigatedTo event is fired. In this event handler, we will initialize the UI controls based on the application settings.

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    //initialize controls in the UI based on oersisted settings
    NightModeCheckbox.IsChecked = appSettings.NightModeSetting;
    if (appSettings.FontSizeSetting == "small")
        FontSizeSmallBtn.IsChecked = true;
    if (appSettings.FontSizeSetting == "medium")
        FontSizeMedBtn.IsChecked = true;
    if (appSettings.FontSizeSetting == "large")
        FontSizeLargeBtn.IsChecked = true;
}
```

### *Using Application Preferences*

Whenever the application main page is loaded, we will use the settings to update the font size and colors of the application.

Open MainPage.xaml and associate a new event handler for page load to the main PhoneApplicationPage. Type space in the Phone:PhoneApplicationPage node and select **Loaded**. Select **<New Event Handler>** which will added the following XAML at the end of Phone:PhoneApplicationPage node and also create a placeholder method in the MainPage class.

```
Loaded="PhoneApplicationPage_Loaded"
```

Just like before, declare AppSettings instance in the MainPage class and instantiate it in the class constructor as shown below.

```
AppSettings appSettings;
// Constructor
public MainPage()
{
    InitializeComponent();
    appSettings = new AppSettings();
}
```

Write the following utility methods and properties in the class. The first method returns a SolidColorBrush from hex values. The other two properties create and return dark and light brushes that will be used to paint the background and foreground.

```
public static SolidColorBrush GetColorFromHexa(string hexaColor)
{
    return new SolidColorBrush(
        Color.FromArgb(
            Convert.ToByte(hexaColor.Substring(1, 2), 16),
            Convert.ToByte(hexaColor.Substring(3, 2), 16),
            Convert.ToByte(hexaColor.Substring(5, 2), 16),
            Convert.ToByte(hexaColor.Substring(7, 2), 16)
        )
    );
}
SolidColorBrush DarkBrush {
    get
    {
        return GetColorFromHexa("#CCDD1167");
    }
}
SolidColorBrush LightBrush
{
    get
    {
        return GetColorFromHexa("#FF00FFFF");
    }
}
```

The following routine returns the size of font based on the FontSize application setting.

```
double AppFontSize
{
    get
    {
        switch (appSettings.FontSizeSetting)
        {
            case "small":
                return 20;
            case "medium":
                return 24;
            case "large":
                return 28;
            default:
                return 24;
        }
    }
}
```

Finally, we will update the font size and the application colors based on the settings. This event handler gets called when the page is loaded which happens when the application is loaded the first time or when we navigate from the settings page.

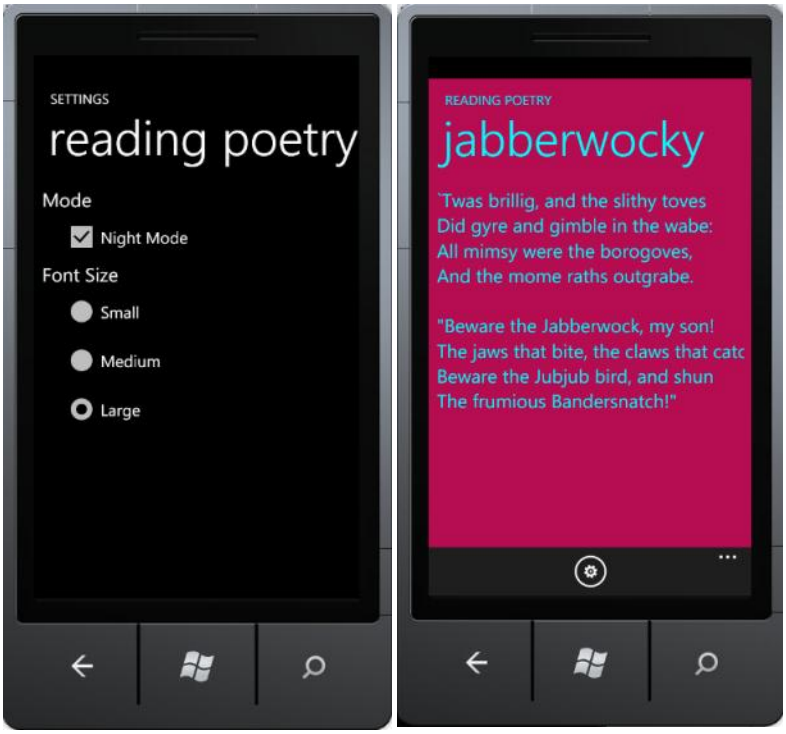
```
private void PhoneApplicationPage_Loaded(object sender, RoutedEventArgs e)
{
```

```
    //Update the fontsize of the poetry textblock.
    PoetryBlock.FontSize = AppFontSize;

    SolidColorBrush bgBrush;
    SolidColorBrush fgBrush;
    if (appSettings.NightModeSetting)
    {
        // use dark brush for the background and light brush for foreground for night mode
        bgBrush = DarkBrush;
        fgBrush = LightBrush;
    }
    else
    {
        // use light brush for the background and dark brush for foreground for day mode
        bgBrush = LightBrush;
        fgBrush = DarkBrush;
    }
    //now paint the entire app layout with background
    LayoutRoot.Background = bgBrush;
    //paint app title, page title and the poetry texblock text with foreground color
    this.Foreground = fgBrush;
    ApplicationTitle.Foreground = fgBrush;
    PageTitle.Foreground = fgBrush;
}
```

Now hit F5 to see the application react to application settings. Here is the behavior with small fonts and day mode.

On the other hand, if you select the night mode and large fonts, it should look like this.




## Conclusions

In contrast to iPhone, Windows Phone 7 requires that the application settings be managed within the application itself. This provides a consistent experience across all third party applications. It also provides an ease of use as the user can change preferences without leaving the application.

While the two platforms differ in their presentation, both platforms use similar mechanism for managing the preferences. They use dictionaries to save preferences as key-value pairs. While iPhone uses application specific file store for persistence, Windows Phone 7 uses IsolatedStorage class to persist preferences in protected storage.

Windows Phone 7 platform provides necessary widgets that closely correspond to widgets used for iPhone application preferences. It is possible to migrate the application preferences from iPhone to Windows Phone 7.

# Chapter 8: Introduction to Windows Phone 7 Notifications for iPhone Developers

## What Are Push Notifications?

Push notifications are used to inform the user of the occurrence of external  events or updates to user's application data.. Consider the following scenario. You are interested in knowing when the price of a particular stock reaches a threshold. Both iPhone and Windows Phone 7 allow only one application to be active in the foreground at any time. You want to know about it even if the application which shows stock prices is not currently running in the foreground. Push notifications allow you to react to events that are of interest to any application, not just the one in the foreground. .

On Windows Phone 7, notifications play a very important and prominent role. The application notifications make the Windows Phone 7 tiles come alive. The notification mechanisms are used to display the most up-to-date status of the application that reflects the events of interest to the user. The tiles work with the notifications to bring information to the user. The user does not have to open the application to the see the updated status.

## Notifications on Windows Phone 7

There are three types of notifications on Windows Phone 7, namely, tile notifications, toast notifications and raw notifications. They serve three distinct purposes.

**Tile notification.** Tile notification is used to show the up-to-date status of the application on the application tile on the start screen of the phone. This is used only if the application is pinned to the start page and is meant to provide information at a glance. There are three types of information included in a tile notification, namely, a picture – for the tile background, a count – for example, think about  the count of email messages and a title that conveys  a message.


**Toast notification**

. A toast notification is a message that pops up on the screen for 10 seconds.  It is meant to provide time sensitive information. It is system wide, but does not disrupt user operation or require user action. If the user taps the notification, the application that sent the notification is launched. The user may dismiss the toast with a flick.



Windows Phone


The toast notification shows a small the application icon on the left with two text items, a title and a subtitle.

**Raw notification**

Unlike toast notifications, raw notifications are in-application notifications that require action. They can be generated by the application itself or sent from a web service. Web service raw notifications only appear within the specified application; there is no system-wide way to display a raw notification. They affect only that application, if the application is not currently running, the raw notification is discarded.

**iPhone and Windows Notifications Compared**

iPhone provides three types of notifications, namely, sound, badge, and message notifications. Badge notifications are used to display a number on the application tile. Message notification is used to display a popup message that user needs to acknowledge. Here is how the notification mechanisms compare on the two platforms.

| Purpose | iPhone | Windows Phone 7 |
|---|---|---|
| Information at a glance | iPhone notification using badge and sound | Live Tile notifications that show a count, a title and a picture |
| Time sensitive information that the user may ignore | N/A | Toast notification with a message |
| Time sensitive information that the user must acknowledge | iPhone notification using message and sound | Raw notifications when application is running |

As compared to the iPhone badge which is used to show an updated count (an application relevant piece of information), WP7 has the additional ability to portray richer information in its tile notifications. With the title message and the updated picture, WP7 notifications make the WP7 tiles come alive.

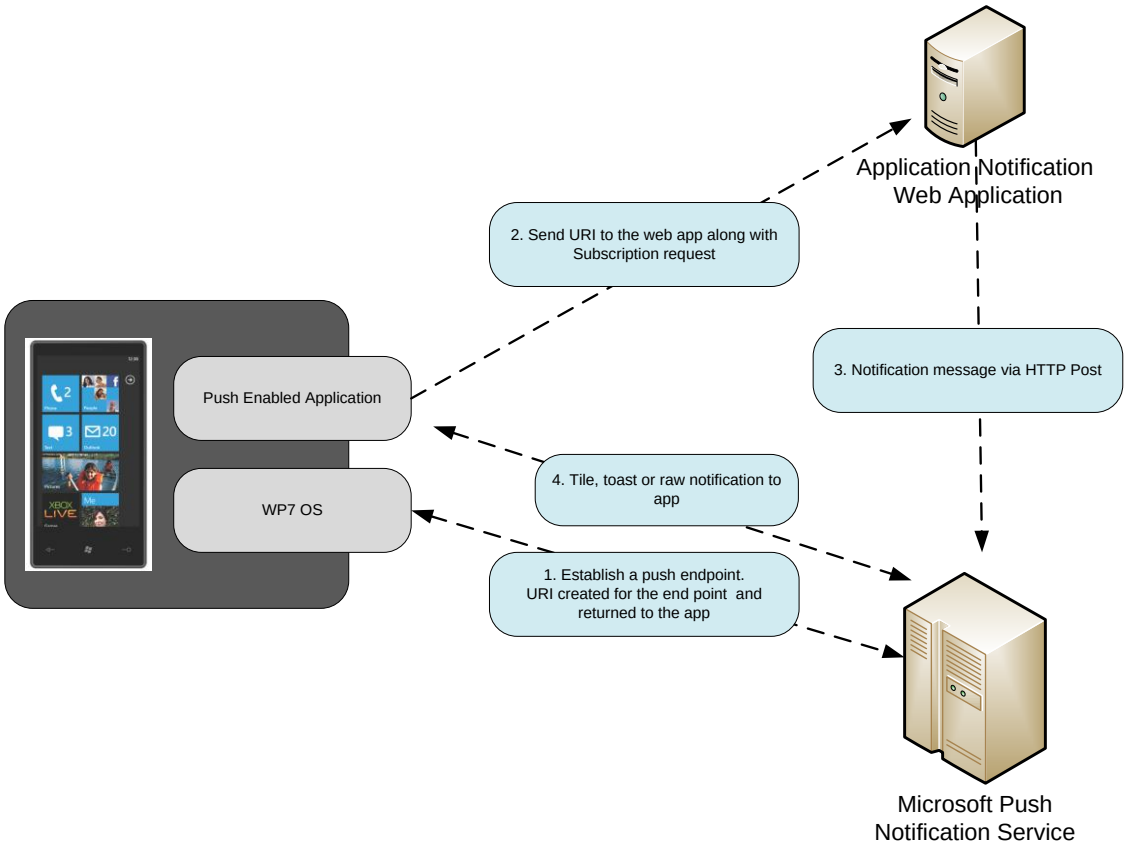
## The Architecture of Windows Phone 7 Push Notifications

At a high level push notifications have a similar architecture on iPhone and Windows Phone 7. There are three parts to the push notification system on both platforms.

1. The phone based application. A user subscribes to events or notifications of interest, either explicitly or implicitly.
2. The web service that sends the notification. The web service monitors external events of interest and checks the application status. It sends a notification to the notification service that relays it to the device.

3. The Apple or Microsoft Push Notification service that pushes the notification to the phone.

The following picture shows the architecture of the Windows Phone 7 Push notifications service.




The phone application with the WP7 OS, the application web service, and the Microsoft Push Notification Service work together to bring the notification to the user.

1. The application uses WP7 to register its notification endpoint with the Microsoft Push Notification Service (MPNS). A new URI is established for the endpoint and returned to the OS which forwards it to the application. Compare this to iPhone where the application creates a unique device ID by registering with the APNS.
2. The application sends the URI and a unique device ID to its notification application web service. This is typically accompanied by the notification subscription request. For example, if the application wants to be notified of changes to a flight schedule, it will send information such as the flight number, along with the notification URI and a unique device ID to the remote application web service. This is similar to what happens on iPhone where the application sends the device token along with the notification subscription.
3. When an event that the user has subscribed to takes place, the web application sends the notification to the MPNS which in turn sends the notification to the device. In response, depending

upon the type of the notification, an action takes place. For iPhone, the notification request is sent to APNS which in turn sends it to the device.

The following table shows the correspondence between parts of the iPhone and Windows Phone 7 notification mechanisms.

| Purpose | iPhone | Windows Phone 7 |
|---|---|---|
| Application registration | Device token | Unique Device ID + URI |
| Subscription notification | Remote application web service | Remote application web service |
| Notification service | Apple Push Notification Service | Microsoft Push Notification Service |

### *Benefits of using WP7 Push Notifications*

There are number of benefits of using WP7 push notifications WP7 uses heartbeat to reduce the number of connections. Additionally, it batches the notifications, which helps reduce the radio usage to conserve the battery.

## Using WP7 Notifications within the Application

Let us look at what it takes to implement push notifications in a WP7 application.

### *Registering for the notifications*

On Windows Phone 7, registering for the notification requires two pieces of information, namely, a unique ID and a URI. This tuple is the counterpart of the device token on iPhone. The URI is used by the MPNS to address the device uniquely. The application push web service sends the notification message that includes the URI to the MPNS, which then forwards it to the device.

Compare this with the registration for notification on iPhone. To receive the notification, you need to download a provisioning profile configured for notification. Once provisioned, the application uses registerForNotificationType using one or more notification types.

**Generating a unique ID**

The device generates its own unique ID using a GUID and saves it in the isolated storage for subsequent runs of the application. The application should subsequently send the same device ID since the notification service uses it to index the subscription information as well as the notification endpoint.

```
//if you have previously created a unique id, use it, otherwise create a new one
if (IsolatedStorageSettings.ApplicationSettings.Contains("DeviceId"))
{
    //retrieve the unique id saved in the isolated storage
    _deviceId = (Guid)IsolatedStorageSettings.ApplicationSettings["DeviceId"];
}
else
{
    //create a new guid and save it in the isolated storage
```

```
    _deviceId = Guid.NewGuid();
    IsolatedStorageSettings.ApplicationSettings["DeviceId"] = _deviceId;
}
// Sset up a notification channel and subscribe to notifications
SetupNotificationChannel();
```

### Obtaining the notification URI

The device registers with MPNS to obtain a URI that MPNS will subsequently use to address the device uniquely. The URI is contained in the `HttpNotificationChannel` (simply called Channel) object. If the Channel with a specific name does not exist, the application creates a new Channel. If the service updates the notification URI, application receives the updated URI via the ChannelUriUpdated event handler.

```
string ChannelName = "MyUniqueChannelName";

private void SetupNotificationChannel()
{
    //Find the channel with the given name
    _channel = HttpNotificationChannel.Find(ChannelName);
    if (_channel == null)
    {
        //Channel does not exist. Create one
        _channel = new HttpNotificationChannel(ChannelName);
        //hookup the event handler to receive updated channel object with the new notification URI
        _channel.ChannelUriUpdated += ChannelUriUpdated;
        _channel.ErrorOccurred += (s, e) => Deployment.Current.Dispatcher.BeginInvoke(() => ErrorOccurred(e));
        //Channel URI will be sent in the event handler after Open is called
        _channel.Open();
    }
    else
    {
        //found an existing channel, now hook up notificaton handlers
        HookupNotificationHandlers();
        //communicate with the app web service for subscribing to notifications
        SubscribeToNotifications();
    }
}
```

### Handling Notifications

Whenever the notification URI is updated, the ChannelUriUpdated event handler is called. Once the application receives the updated URI, it binds the channel to the toast and tile notifications if they are not bound already. These binding are used when the application is not running. WP7 will handle these notifications in the standard manner as described above.  The iPhone too provides a standard manner in which notifications are handled when the application is not running. This behavior is configured automatically on the iPhone.

```
private void ChannelUriUpdated(object sender, NotificationChannelUriEventArgs e)
{
    //retrieve the Channel again as the URI is updated
    _channel = HttpNotificationChannel.Find(ChannelName);
    //bind the channel with Toast and Tile
    if (!_channel.IsShellToastBound)
    {
        _channel.BindToShellToast();
    }
    if (!_channel.IsShellTileBound)
```

```
    {
        _channel.BindToShellTile();
    }
    //Setup the notification event handlers
    HookupNotificationHandlers();
    //Send the device id and the URI to the app notification web service.
    //web service should update the URI if it is already stored in the web service
    SubscribeToNotifications();
}
```

The following method shows how the toast and raw notifications are handled when the application is running. When these notifications are received, the ToastReceived and HttpNotificationReceived methods are invoked. ToastReceived is used for toast notifications whereas HttpNotificationReceived is used for raw notifications. A tile notification has a standard behavior and a running application is not notified. Compare that with iPhone where the application receives didReceiveRemoteNotification call back if the application is running when the notification arrives. The application can then handle the notifications the way it wants.

Since these notifications are not received on the UI thread, we use BeginInvoke to handle them on the UI thread. In this example, these notifications are displayed on the page but you may choose to use them in other ways.

```
private void HookupNotificationHandlers()
{

    _channel.ShellToastNotificationReceived += (s, e) => Deployment.Current.Dispatcher.BeginInvoke(() =>
ToastReceived(e));
    _channel.HttpNotificationReceived += (s, e) => Deployment.Current.Dispatcher.BeginInvoke(() =>
HttpNotificationReceived(e));
    _channel.ErrorOccurred += (s, e) => Deployment.Current.Dispatcher.BeginInvoke(() => ErrorOccurred(e));
}

private void HttpNotificationReceived(HttpNotificationEventArgs e)
{
    var reader = new StreamReader(e.Notification.Body);
    var message = reader.ReadToEnd();
    notifications.Items.Add("Raw notification message : " + message);
    reader.Close();
}

private void ToastReceived(NotificationEventArgs e)
{
    notifications.Items.Add("Toast notification message : " + e.Collection["wp:Text1"]);
}
```

### Subscribing to Notifications

Once the application has the updated URI, it can use it to subscribe to notifications with its remote application web service. In our sample application, we will print the channel URI to the console and use it from our message sender application. You should send both the device ID and the URI along with additional parameters needed to subscribe to application specific notifications.

```
private void SubscribeToNotifications()
{
    Debug.WriteLine("Use this channel URI to send notification:" + _channel.ChannelUri.ToString());
```

```
}
```

### *Sending notifications*

In our sample example, we demonstrate how  to send the notification using a simple WPF application. It takes the URI and other parameters needed for each type of notification and performs HTTP post to MPNS.

As opposed to the JSON format used in iPhone, WP7 notification uses XML format for the tile and toast messages. In this example, the first three methods format the message appropriately and call the SendMessage method. The structure of the three methods is very similar. Messages are sent to the device using HTTP POST to the application URI received in the subscription request.  In this example, we ignore the status of the notification POST but ideally, you should handle the response appropriately.

```
public void SendToastNotification(string message)
{
    string toastMessage = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
         "<wp:Notification xmlns:wp=\"WPNotification\">" +
            "<wp:Toast>" +
               "<wp:Text1>{0}</wp:Text1>" +
              "</wp:Toast>" +
         "</wp:Notification>";
    string formattedToastMessage = string.Format(toastMessage, message);
    byte[] messageBytes = System.Text.Encoding.UTF8.GetBytes(formattedToastMessage);
    SendMessage(uri, messageBytes, NotificationType.Toast);
}
public void SendRawNotification(string message)
{
    byte[] messageBytes = Encoding.UTF8.GetBytes(message);
    SendMessage(uri, messageBytes, NotificationType.Raw);
}
public void SendTileUpdate(string title, int count, string imageUrl)
{
    string tileMessage = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
        "<wp:Notification xmlns:wp=\"WPNotification\">" +
           "<wp:Tile>" +
              "<wp:BackgroundImage>{0}</wp:BackgroundImage>" +
              "<wp:Count>{1}</wp:Count>" +
              "<wp:Title>{2}</wp:Title>" +
           "</wp:Tile> " +
        "</wp:Notification>";
    string formattedTileMessage = string.Format(tileMessage, imageUrl, count, title);
    byte[] messageBytes = System.Text.Encoding.UTF8.GetBytes(formattedTileMessage);
    SendMessage(uri, messageBytes, NotificationType.Tile);
}

private void SendMessage(Uri uri, byte[] messageBytes, NotificationType notificationType)
{
    var request = (HttpWebRequest)WebRequest.Create(uri);
    request.Method = WebRequestMethods.Http.Post;
    request.ContentType = "text/xml";
    request.ContentLength = messageBytes.Length;
    request.Headers.Add("X-MessageID", Guid.NewGuid().ToString());
    switch (notificationType)
    {
        case NotificationType.Toast:
            request.Headers["X-WindowsPhone-Target"] = "toast";
            request.Headers.Add("X-NotificationClass", ((int)BatchingInterval.ToastImmediately).ToString());
            break;
        case NotificationType.Tile:
            request.Headers["X-WindowsPhone-Target"] = "token";
            request.Headers.Add("X-NotificationClass", ((int)BatchingInterval.TileImmediately).ToString());
            break;
        case NotificationType.Raw:
            request.Headers.Add("X-NotificationClass", ((int)BatchingInterval.RawImmediately).ToString());
            break;
```

```
    }
    using (var requestStream = request.GetRequestStream())
    {
        requestStream.Write(messageBytes, 0, messageBytes.Length);
    }

}
```

**Sending notifications from the application web service**

In a real  WP7 application, you will send both the device ID and the Channel URI to the application web service. The web service should store the Channel URIs indexed by the device ID. While the  channel URI may change, the device ID remains the same. The web service should update the channel URI every time it receives the device ID and channel URI from the device.

## Throttling

MPNS allows both unauthenticated and authenticated notifications. However, unauthenticated push requests are throttled if they exceed 500 messages per day. There are no such restrictions on authenticated requests which uses client SSL for authentication.

**Comparing iPhone and WP7 Notifications**

While the two services are similar, there are subtle differences between iPhone and WP7 notification mechanisms. Developers can take advantage of the MPNS mechanism to improve the user experience.

On Windows Phone 7, tile notification can use remote image files which can be used to show fresh status on WP7 tiles. Similarly on WP7, there is no limit on the size of the notification messages and developers can use them creatively. There is no guarantee of delivery of messages on either service. Notifications may not be delivered MPNS provides a status of the delivery which developers can use to resend the notification.

| Functionality | iPhone | Windows Phone7 |
|---|---|---|
| Notification Service | Apple Push Notification Service | Microsoft Notification Service |
| Payload format | JSON | XML |
| Payload resource | Local sound files | Local or remote image files. Size of remote file can be upto 80kb |
| Payload length | 256 bytes | No limit on the size of payload |
| Delivery guarantee | No guarantee | No guarantee |
| Delivery class | Delivered ASAP | Delivered immediately or within 450 or 900 seconds |
| Delivery status | No status of delivery | Able to get the status of delivery. |
| Connection Trust | TLS between application web service and APNS | No SSL for < 500 notifications or SSL otherwise for connection between app web service and MPNS |

## Summary

The push notification mechanism plays a very critical role in Windows Phone 7. It makes tiles come alive and provides up-to-date information at a glance. Windows Phone 7 provides a compelling notification mechanism that is easy to use. Developers can start using notifications immediately even without setting up client authentication. While the architecture of the services on iPhone and WP7 is similar, developers can take advantage of the additional features WP7 notification mechanism provides.

## Resources

1. Push Notifications for Windows Phone: MSDN reference on Windows Phone push notification.
2. Windows Phone 7 in 7: Push Notifications

# Next Chapters [list TBD]

## Coming soon



Windows Phone