UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria dell'Informazione

USER INTERFACES IN iOS AND WINDOWS PHONE: A COMPARISON

RELATORE:

Prof. Carlo Fantozzi

TESI DI LAUREA DI:

Silvia Gandin

Anno Accademico 2012/2013

Index

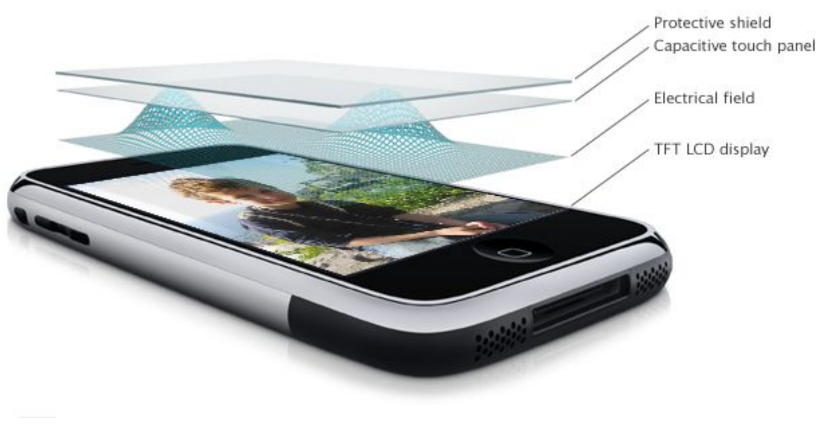
1. INTRODUCTION

1.1. AIM OF THE ESSAY

The announcement of Windows Phone 8 in November 2012 launched a challenge to the well-set duo Android/iOS, the leading powers in mobile operating systems. Windows Phone user interface has surely impressed the competitors, for its fresh and dynamic look. The question is if there is something more than this new design, because it is certainly not enough to worry the two world leaders that set the rules of the mobile market. It was clear that Windows Phone 7 was not as mature as iOS or Android - multitasking was very limited, the apps were not there, and so on - but the last version, codenamed Apollo, promises great improvements.
This term paper contains a comparison between iOS and Windows Phone, regarding especially the user interface, that rules the interaction between the user itself and the device.
Chapter 1 deals with a brief description of the evolution of the two platforms from their first appearance to the present.
In Chapter 2, the user interface principles and guidelines are put into comparison, with a deeper analysis on the Windows Phone ones.
Chapter 3 carries out a technical examination of the management of various user interface problems, like different screen resolutions, orientation changes, and so on. Some Windows Phone main elements are described in detail.

1.2. THE EVOLUTION OF iOS

The iPhone was shown to the world by Apple's CEO Steve Jobs on January 10, 2007. Since then, the world of mobile computing has known a revolutionary change, and it was Apple which set the rules for all the other competitors.
Despite many similarities – like running on the same Unix core, this new OS and Mac OS X were clearly different, as the former was created to work specifically on mobile devices. Although it is now referred to as iOS, when it was first introduced it was called "iPhone OS", and changed its name only with its fourth major release, in June 2010.
The original iPhone represented a real revolution, although it was not above competitors considering only the standard set of features. The then established systems Windows Mobile, Palm OS, Symbian, and BlackBerry presented in fact a wide range of features that iPhone did not have. The Apple's device did not support 3G, multitasking, MMS, and 3rd party apps; users could not copy or paste text, attach arbitrary files to emails, or customize the home screen, and it hid the filesystem, closing the access to hackers and developers.
Despite the fact that it lacked all these features, the revolution of iOS was that it greatly focused on the user experience. Among the various characteristics in iOS 1.0, two of them were particularly innovative for the mobile world: the user interface, and Mobile Safari Web Browser.
Before the introduction of the iPhone, smartphone screens were not touch, or used a resistive technology which often frustrated users, forcing them to use a stylus for more accuracy and slowing down the gestures. The iPhone changed everything with its capacitive fast and smooth touchscreen, together with a simple but powerful user interaction model. For the first time, a device presented only five physical buttons, making touch the primary interaction model.



The iPhone touchscreen

The speed and directness in iOS 1.0 was something new which contributed to attract people's interest all over the world.
Those new gestures were implemented perfectly in the Safari web browser. Steve Jobs was right when he announced that it was well above any competition.

Although it had received a lot of criticism for not supporting the Flash plugin, it was the first mobile web browser to offer an experience almost identical to that of a full desktop browser. Unlike other mobile web browsers, Safari presented the web fully, without the formatting problems that were common at the time.
After this revolutionary start, Apple continued meeting success by updating iOS and presenting new devices, adding gradually what users desired and much more.

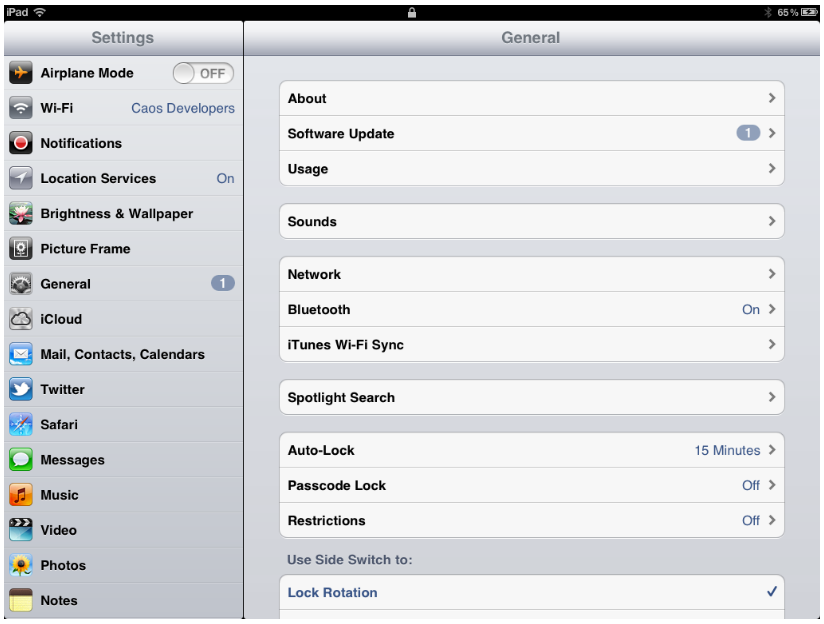
With iOS 2 in July 2008, Apple introduced the App Store for third-party apps, together with the iOS SDK, letting developers build applications in such a simple and clean way that has no equal to the other platforms. The market started a huge spread, giving users so many functional, well-designed, and advanced apps, that still nowadays it is unrivaled.
However, the introduction of the App Store was followed by some criticism, not ended even today. The reason is that Apple has not completely opened up iOS, both by hiding the file system to users, and by enforcing a strict policy for publishing apps. After it is developed, an app must be sent to Apple for approval, with lots of limitations in terms of modifying system settings.

IOS 3 and the iPhone 3GS were released in June 2009. Like the 3GS, iOS 3 did not have major new features, as Apple focused on adding a huge list of little improvements to the operating system.
A further big step on Apple's race towards success was the iPad, launched in April 2010.

iPhone 4 and iPad

The first iPad met with an extraordinary success. Competitors were urged to start developing tablets, because the iPad set new needs on people, becoming one of the most desired devices.
By bringing iOS to the iPad, Apple did not just expand the iPhone UI, but it designed different views to adapt to the larger screen. For example, when a list of

items is presented, the list is shown on a left-hand sidebar, while the selected item is displayed on the right.


iPad 65 %

Settings

Airplane Mode OFF
Wi-Fi Caos Developers
Notifications
Location Services On
Brightness & Wallpaper
Picture Frame
General 1
iCloud
Mail, Contacts, Calendars
Twitter
Safari
Messages
Music
Video
Photos
Notes

General

About
Software Update 1
Usage

Sounds

Network
Bluetooth On
iTunes Wi-Fi Sync

Spotlight Search

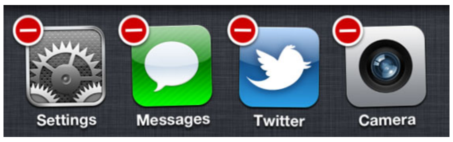
Auto-Lock 15 Minutes
Passcode Lock Off
Restrictions Off

Use Side Switch to:

Lock Rotation ✓

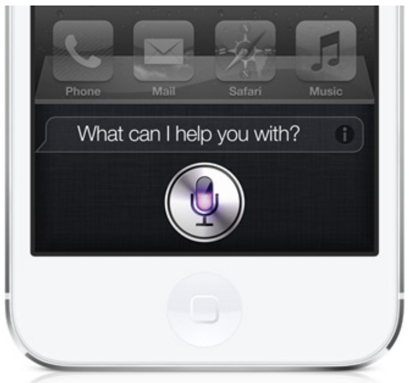
iPad UI

IOS 4 came alongside the iPhone 4, bringing FaceTime video chat, which takes advantage of the new spectacular Retina Display, and finally multitasking.



Multitasking

Further steps were made with iOS 5 in October 2011, introducing a lot of new features. The most innovative was Siri, the first virtual assistant on a mobile phone, that gives users nearly the illusion of interacting with a living being. Siri has been developed further after iOS 5, becoming more and more usable and capable to adapt to plenty of circumstances. Users can make reservations for dinner, check local movie listings, get sports scores, post updates to Facebook and Twitter, among plenty of other actions.

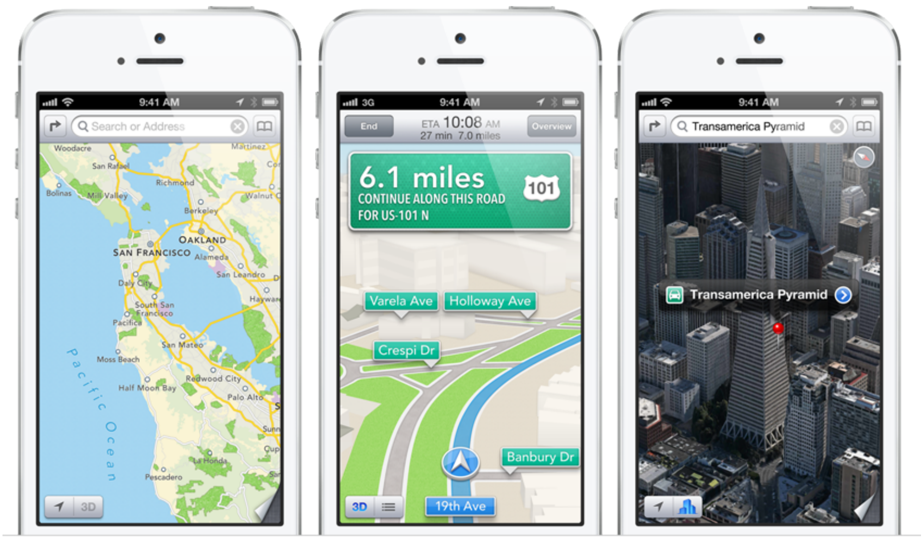


What can I help you with?

Siri virtual assistant

Not less important was iCloud, the new Apple cloud service. It was certainly not the first one, but its simplicity and usefulness was appreciated by users, forcing competitors to find similar solutions. Another great addition was Notification Center, that groups all notifications from apps together, and iMessage, Apple's own system for sending free short messages.

Finally, Apple released iOS 6 on June 11th, 2012. This is the latest update so far, and although it has not upset critics, there are few new interesting features, together with lots of little improvements. Maps are completely revamped, breaking the long-last relationship with Google. A notable addition is a turn-by-turn navigation function with spoken directions. The new app has some other promising features, but the huge quantity of errors shows that it is not yet ready for the market, causing hard criticism from users.

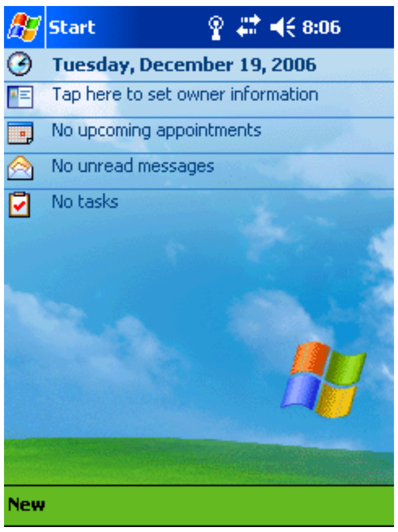


iOS 6 new Maps

Apart from this major change, iOS 6 is not much different from iOS 5, showing that Apple is more concerned about refining its OS than about taking risks and pushing forward.

1.3. THE EVOLUTION OF MICROSOFT MOBILE OSes

Microsoft's history regarding mobile operating systems started with Pocket PC in 2000, but this OS was used for smartphones only in 2002. One year later Windows Mobile came out, lasted for nearly 7 years through various updates. It is based on the Windows CE kernel and designed to have features and appearance somewhat similar to desktop versions of Windows.



Tuesday, December 19, 2006
Tap here to set owner information
No upcoming appointments
No unread messages
No tasks

Windows Mobile 2003 for Pocket PC

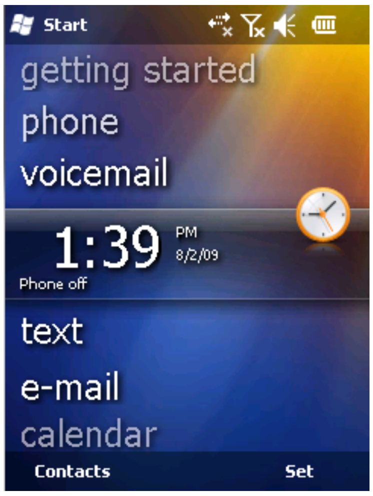
Windows Phone is supplied with a suite of basic applications developed with the Microsoft Windows API, like Office, Internet Explorer, Windows Media Player and Outlook.
Third parties can develop software for Windows Mobile with fewer restrictions than those imposed by Apple. Software applications were purchasable from Windows Marketplace for Mobile. Most early devices came with a stylus, which can be used to enter commands by tapping on the screen. The primary touch input technology behind most devices was resistive touchscreen, but some devices featured also slide-out keyboards.

There were three main versions of Windows Mobile:

- Windows Mobile Professional for smartphones with touchscreens;
- Windows Mobile Standard for mobile phones without touchscreens;
- Windows Mobile Classic for personal digital assistants or Pocket PCs.

HTC was the main producer of Windows Mobile devices, and built almost 80% of the 50 million sold from Windows Mobile launch until February 2009.

The last version of Windows Mobile is the 6.5, presented in February 2009.



Windows Mobile 6.5

Windows Mobile 6.5 was not in the plans of Microsoft, and its chief executive, Steve Ballmer, described it as "not the full release [Microsoft] wanted".
In fact Microsoft began working on a major update codenamed "Photon" from 2004, but works proceeded slowly, causing the cancellation of the project. Only in 2008 did Microsoft manage to restart the development of a completely new mobile operating system. Several delays urged Microsoft to release firstly Windows Mobile 6.5, that was followed only by three minor updates, made to satisfy consumers during the transition period. In fact works fully shifted from Windows Mobile to its successor Windows Phone, causing the new OS's lack of support for Windows Mobile applications and devices designed for the older OS.

The new operating system was first known as Windows Phone 7 Series, and was announced at Mobile World Congress in Barcelona on February 15, 2010. After some criticism for the long name, it was officially shortened to just Windows Phone 7.

Windows Phone 7 Start Screen

Unlike its predecessor, Windows Phone is primarily aimed at the consumer market rather than the enterprise one. This implies to a totally new user interface, commonly referred to as Metro, with its famous Live Tiles.
Windows Phone 7 sets rigid rules on hardware requirements, so as to provide a consistent user experience. The most important requirements are the presence of a capacitive 4-point multi-touch screen with WVGA (480x800) resolution, 512 MB of RAM (lowered later), six dedicated hardware buttons (back, Start, search, 2-stage camera, power/sleep and volume buttons), and the basic sensors, such as the accelerometer, the ambient light sensor, the proximity sensor and the one for Assisted GPS.
However, Windows Phone 7 was not at the same level as iOS or Android, missing some must-have features. Windows Phone 7.5 (May 2011), codenamed Mango, tried to fill the gap by adding a mobile version of Internet Explorer 9, multitasking for third-party apps, tethering connection, Twitter integration and much more. The last update for the "7.x series" is 7.8, released in January 2013, and brings some graphical add-ons introduced with Windows Phone 8.
Although it is an interesting OS with some innovative aspects like the user interface, Windows Phone 7 has not met great success, managing to capture only about 3% of the smartphone market share. One reason may be that the hardware is decent but not superlative, as it is the same of many existing Android devices. Others believe that Microsoft has adopted a weak marketing strategy, not exciting people at all.
Trying to learn from its mistakes, Microsoft studied the next step well. During the works for the new OS, the company did not let much information pass to the world outside, creating a fair sense of expectation.

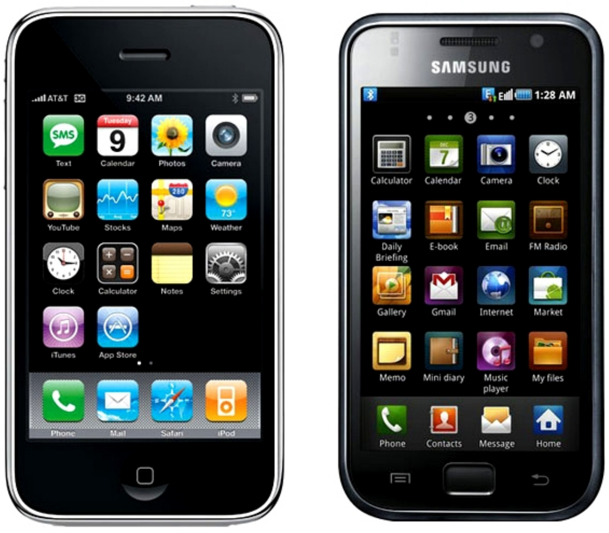
On June 20, 2012, Microsoft unveiled Windows Phone 8, and released it to consumers on October 29.

Windows Phone 8 Start Screen

Like in the past with Windows Mobile, Windows Phone 7.x devices cannot be updated to the new operating system, and new applications compiled specifically for Windows Phone 8 are not available for them.
Windows Phone 8 brings lots of innovations, becoming a more mature operating system that can compete with the two dominant corporations, Apple and Google. First of all, it now runs on the Windows NT kernel, the same as Windows 8, allowing applications to be easily ported between the two platforms. It also supports devices with larger screens and multi-core processors, near-field communication (NFC), removable storage, has backwards compatibility with Windows Phone 7 apps, a redesigned home screen, and more.

1.4. PATENT WAR

In these years Apple has undoubtedly set the standards, forcing competitors to keep up and not lose ground. Many companies like RIM have lost the match, and only Android manages to fight well against iOS.
This battle has soon grown a legal one, becoming the so called "patent war". In fact Apple has patented many features of its devices, from the design to the gestures. Google has mainly copied some of them, adding things that users complain the iPhone misses, like a more open and liberal operating system, widgets, customization and so on. However, it is clear to everyone that there are many devices so similar to the iPhone that you can almost call them "clones", not only for the design, but also for the user interface.



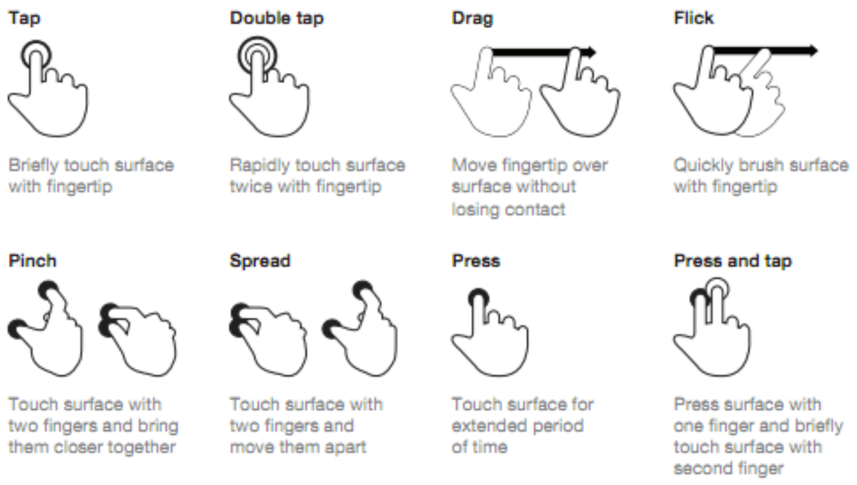
Similarities between iPhone 3GS and a Samsung smartphone

Finally, in August 2012, a US jury declared that Samsung, the main manufacturer of Android's smartphones, had intentionally copied Apple, and had to pay $1.05 billion in damages. The patent war has not ended there, trials continue all over the world, but this defeat has shown that it is time to open new roads, to explore different solutions. Lots of people continue to express criticism about the limitations set by Apple's patents, complaining that they would block competition. Others say that this will lead instead to great innovations, forcing companies to imagine other ways for doing and showing things, maybe finding better ones.
So the patent war would not kill competition: it would only make competition better, pushing companies to innovate, like it is already happening. And the patent war will not allow Apple to rest on its past success: instead it will force it to maintain its power with further innovations.
An example of how good this can be is Microsoft. Microsoft realized that it was worthless to clone the iPhone, and that it needed to do something new. So, it has developed Windows Phone, trying to change the way users do things, but at the same time to create an intuitive and easy-to-use interface.

2. A FUNCTIONAL COMPARISON

2.1. iOS

In the iOS Human Interface Guidelines *[Bibliography, (7)]*, Apple dedicates a lot of attention to describe how the user experience should be. The key point is the Multi-Touch screen, the innovative feature that set the success of the first iPhone. In fact, it determined a new way of seeing a device: the display became a sort of window through whom users physically interact with the content. Removing any intermediary medium such as a mouse, direct manipulation of objects on screen gives people a hight sense of control over them. For example, to turn a page in iBook app the user has to do it almost literally, "taking" the border of the page and moving it onto the other side.
Gestures are the most direct method for interacting with the device, and people have become used to the standard set of them provided by Apple. Because they are consistently present in the built-in apps, users expect to use them in all the other apps. Tap, swipe, and pinch-to-zoom are so familiar that there is no need of instructions on how to do basic operations.


Tap
Briefly touch surface with fingertip
Double tap
Rapidly touch surface twice with fingertip
Drag
Move fingertip over surface without losing contact
Flick
Quickly brush surface with fingertip
Pinch
Touch surface with two fingers and bring them closer together
Spread
Touch surface with two fingers and move them apart
Press
Touch surface for extended period of time
Press and tap
Press surface with one finger and briefly touch surface with second finger

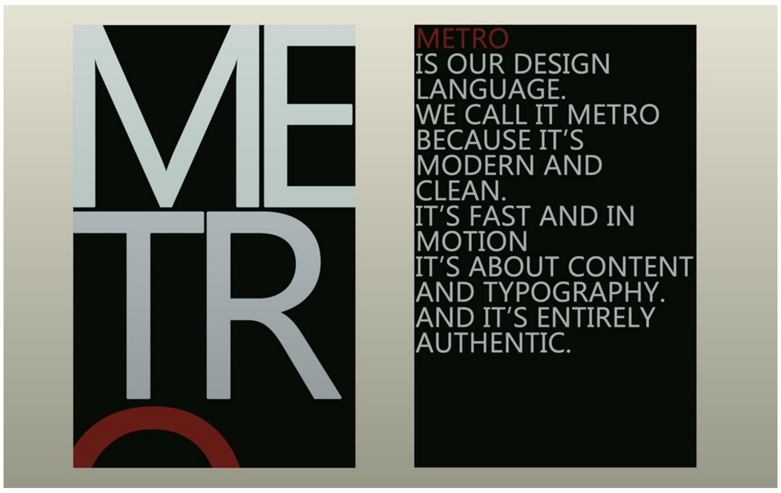
Standard iOS gestures

This is definitely an element that marks iOS user interface even nowadays, making it simple and intuitive to all kind of users. For the same reason, Apple suggests developers to create clear and easy-to-use apps, avoiding too much instructions on screen and using consistency. Consistency is a fundamental principle in Apple's outlook, and states that developers' apps should behave as closer as possible to the apps provided by the platform vendor. To achieve this, Apple suggests using the standard set of controls, views and gestures. An app should be consistent also within itself, so as not to confuse the user. Therefore consistency contributes to maintaning the user experience as simple as possible, uniforming paradigms and actions.

Another way to interact with the device in addition to gestures is voice. Vocal input has become a real competitor to touch input with the introduction of Siri. It, or better she, as Apple calls this feature, is like a real personal assistant, and executes lots of actions following the user's voice commands. This creates a bond between the user and the device, that is seen less and less like a machine, and more and more like an always-available companion.

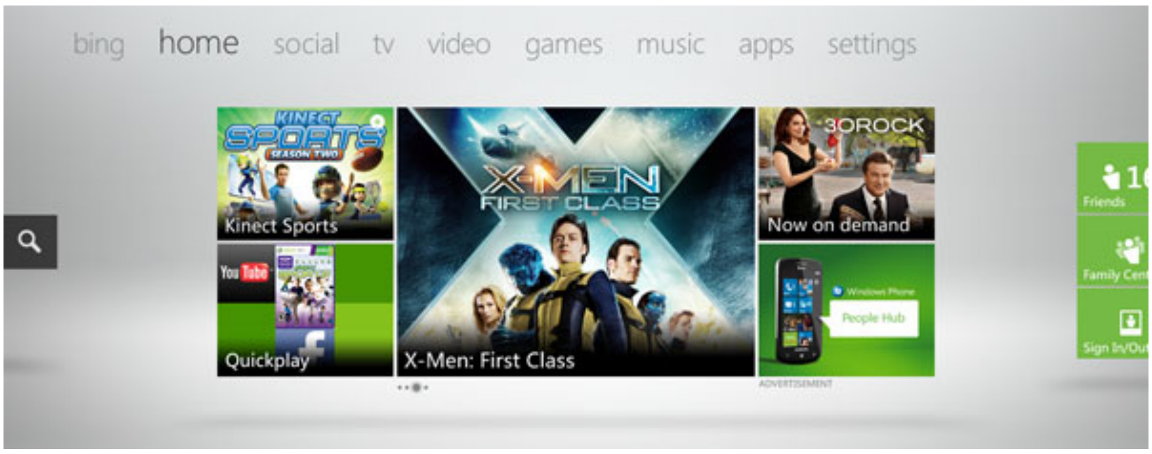
2.2. WINDOWS PHONE

The most notable difference between Windows Phone and iOS is the user interface. Microsoft has worked a lot on it, trying to impress people with something completely new. The key word is undoubtedly "live": everything is up-to-date, vital, in a continuos flux of information. The so-called Start Screen is made of Live Tiles, colored squares of various dimensions that replace the regular and symmetric icons of iOS home screen, called Springboard.
Microsoft user interface is known as Metro UI, although it has recently changed its name to Modern UI, due to legal issues. The principle of the Metro UI takes inspiration from the Swiss Movement of the 1960s, which meant to "communicate to people through design, while being different yet direct". It is also partly inspired by the clear and effective signs commonly found in public transport systems.


METRO
METRO
IS OUR DESIGN LANGUAGE.
WE CALL IT METRO BECAUSE IT'S MODERN AND CLEAN.
IT'S FAST AND IN MOTION
IT'S ABOUT CONTENT AND TYPOGRAPHY.
AND IT'S ENTIRELY AUTHENTIC.

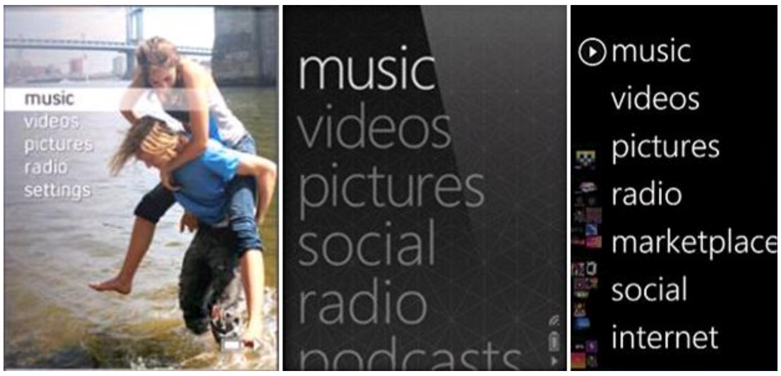
Metro design manifest

The Metro design was used for the first time in Windows XP Media Center product in 2005. Microsoft tried to create an interface based more on typography and less on chrome and buttons. Microsoft also progressively started to redesign the Xbox 360 interface into a more Metro-like design.



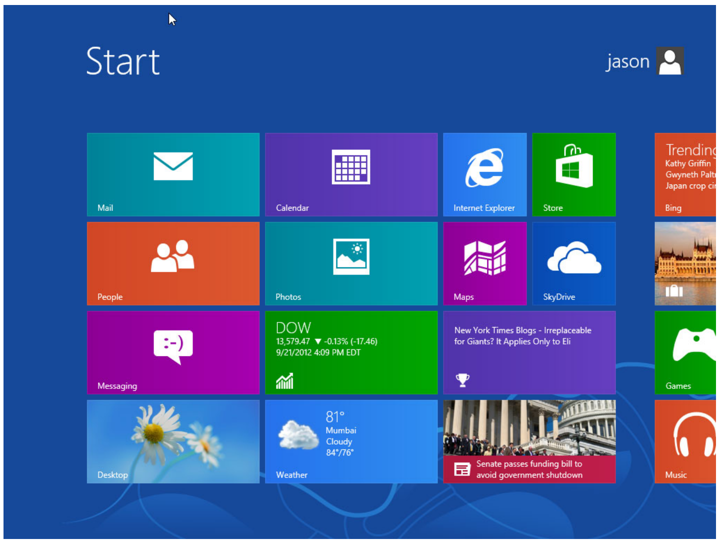
Xbox 360 Dashboard

But the real forerunner of Windows Phone were definitely Zune devices. In 2006 the Zune 30 device was released and belonged to the early generation of music players. Next generations that followed up, especially Zune HD, increasingly featured Metro interface, designating it as Microsoft official design language.



Zune HD interface

Both the new operating systems, Windows Phone 8 and Windows 8, fully deploy Metro design, and especially for the last one it is a great change. Its new Start Screen resembles the home screen of Windows Phone. For the first time the classic desktop is not the primary user interface any more, although it is available through a tile.



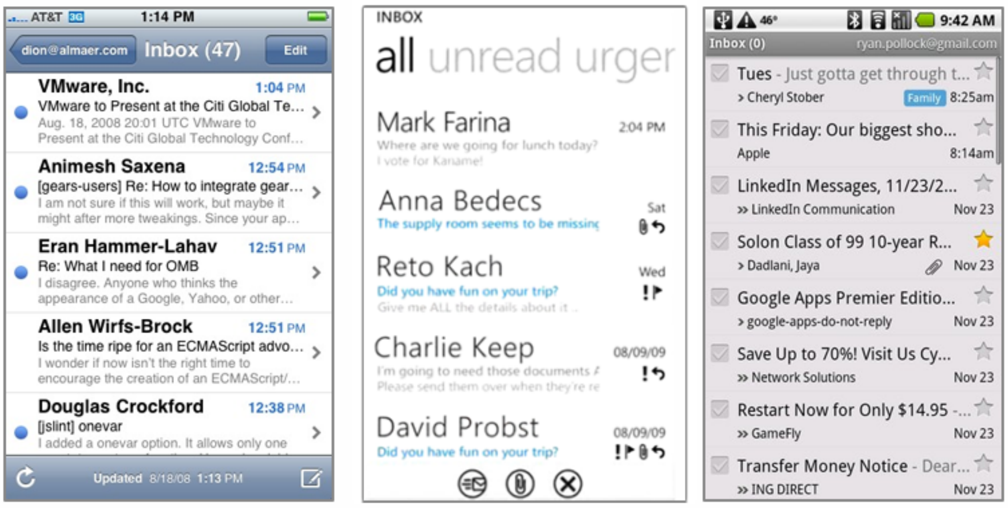
New York Times Blogs - Irreplaceable for Giants? It Applies Only to Eli

Senate passes funding bill to avoid government shutdown

Windows 8 Start Screen

Microsoft wished its new products to stand out from the other mobile platforms, which all followed a general template of having pages of icons and perhaps some widgets. Microsoft wanted a simple but iconic UI. By deleting all superfluous elements, the interface becomes more clear and polished, unique. Instead of being just the way to reach content, the interface shows it directly.
The key design principle of Metro is therefore better focused on the data of applications, counting on typography more than on graphics, "content before chrome". The results are large texts that catch the eye, in a modified version of the Segoe font, which was inspired by Helvetica and the Swiss Movement.

For example, font is used in different sizes and weights to convey structured information. Not all data are the same, and through typography their importance is shown clearly. For this reason bullet points are unnecessary, and therefore not used in Metro.
Consequently, Metro design has to follow a strict organization. Missing almost every graphical layout element, the content grid is made up of only the content itself. So, the designer has to select carefully the elements on which to build the content path, in a hierarchy based on importance.
If wrong elements are chosen, the content will not be easily reachable by the user. Differences with iOS, or worse with Android, are evident in this screenshots.


AT&T 3G 1:14 PM
dion@almaer.com Inbox (47) Edit

VMware, Inc. 1:04 PM
VMware to Present at the Citi Global Te…
Aug. 18, 2008 20:01 UTC VMware to Present at the Citi Global Technology Conf…

Animesh Saxena 12:54 PM
[gears-users] Re: How to integrate gear…
I am not sure if this will work, but maybe it might after more tweakings. Since your ap…

Eran Hammer-Lahav 12:51 PM
Re: What I need for OMB
I disagree. Anyone who thinks the appearance of a Google, Yahoo, or other…

Allen Wirfs-Brock 12:51 PM
Is the time ripe for an ECMAScript advo…
I wonder if now isn't the right time to encourage the creation of an ECMAScript/…

Douglas Crockford 12:38 PM
[jslint] onevar
I added a onevar option. It allows only one

Updated 8/18/08 1:13 PM

INBOX
all unread urger

Mark Farina 2:04 PM
Where are we going for lunch today?
I vote for Kaname!

Anna Bedecs Sat
The supply room seems to be missinc

Reto Kach Wed
Did you have fun on your trip?
Give me ALL the details about it …

Charlie Keep 08/09/09
I'm going to need those documents /
Please send them over when they're re

David Probst 08/09/09
Did you have fun on your trip?

46° 9:42 AM
Inbox (0) ryan.pollock@gmail.com

Tues - Just gotta get through t…
> Cheryl Stober Family 8:25am

This Friday: Our biggest sho…
Apple 8:14am

LinkedIn Messages, 11/23/2…
» LinkedIn Communication Nov 23

Solon Class of 99 10-year R…
> Dadlani, Jaya Nov 23

Google Apps Premier Editio…
> google-apps-do-not-reply Nov 23

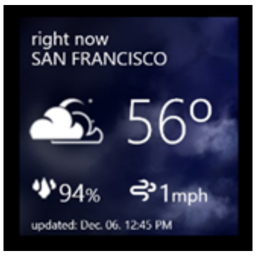
Save Up to 70%! Visit Us Cy…
» Network Solutions Nov 23

Restart Now for Only $14.95 -…
» GameFly Nov 23

Transfer Money Notice - Dear…
» ING DIRECT Nov 23

Mail app in iOS, Windows Phone 8 and Android

The key element of Windows Phone interface is the Live Tile, which embodies all these principles. It is a visual shortcut for an app or its content that users can set in the Start view, in a procedure called "pinning". A Live Tile is something more than an icon, because it can show information in real time – from emails to photos to weather to travel progress – without having to click on applications. For instance, weather forecasts can be shown dynamically on a dedicated tile.

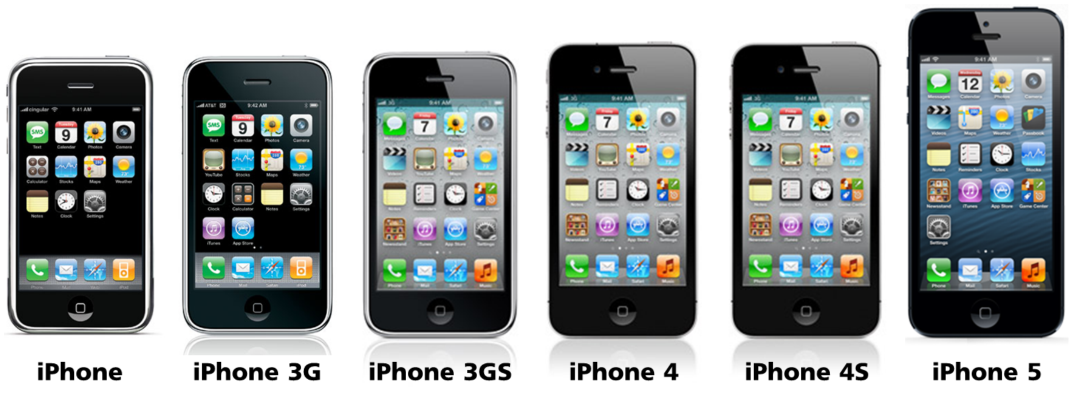


A weather Live Tile

Live Tiles are a new alternative to iOS static icons and Android big intrusive widgets. Comparing to iOS, they perfectly show the new principles under the user interface: liveliness and motion. While the only change in iOS icons are small counter badges, Live Tiles bring plenty of information, without the need of any direct user interaction.

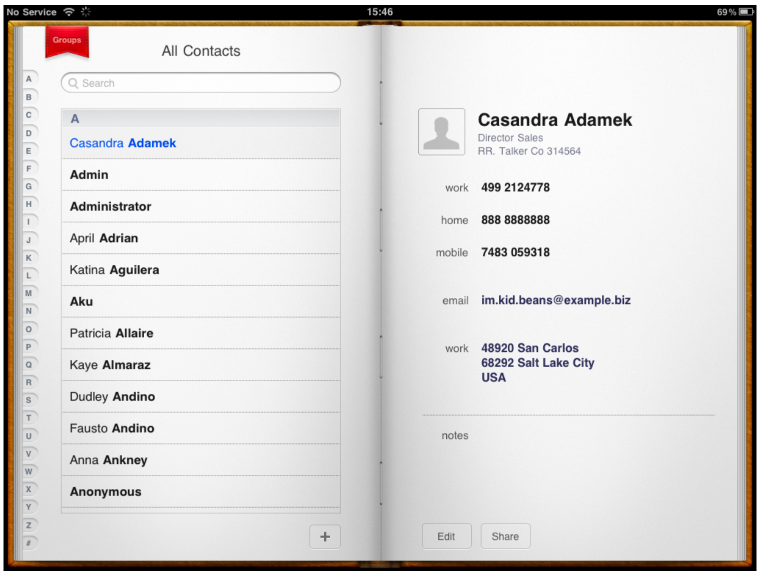
2.3. FURTHER DIFFERENCES

The launch of Windows Phone 8 should concern Apple, because users begin to get tired of the same static Springboard. Although it works well, is simple and polished following Apple design principles, it has remained almost unchanged since the launch of the first iPhone.



Springboard evolution in all iPhone models

The Springboard exhibits pages of icons, seen by swiping horizontally, and a fixed dock at the bottom of the screen, with four main icons. Users can change their positioning, and group apps into folders, but here the customization ends. The difference with the continuos-changing and customizable Live Tiles is evident.
Besides the contrast between the static nature and vitality of the two user interfaces, another important difference is that Metro is “authentically digital”, in contrast to Apple's skeuomorphic view. Skeuomorphism, as applied in digital interfaces, aims at designing elements looking like real-world objects. An exaustive example is provided by Apple, that designed its iBook, Calendar, Notes and Reminders apps to look like their physical counterparts.



A
Casandra Adamek
Admin
Administrator
April Adrian
Katina Aguilera
Aku
Patricia Allaire
Kaye Almaraz
Dudley Andino
Fausto Andino
Anna Ankney
Anonymous

Casandra Adamek
Director Sales
RR. Talker Co 314564

work 499 2124778
home 888 8888888
mobile 7483 059318
email im.kid.beans@example.biz
work 48920 San Carlos
68292 Salt Lake City
USA
notes

Contacts app on iPad

Other metaphors in iOS are the sliding on-off switcher, that resembles the real object, or the spinning picker wheel for selecting the date.
But there are lots of other examples in many digital applications. The shutter clicking sound that camera apps make when taking a photo is totally fake, there's no real mechanical movement that makes it. And the little bar under the F and J keys, on the iPad keyboard, doesn't help blind users like in the real keyboard, because of course the touchscreen lacks protrusions.
Apple says that "the more true to life your application looks and behaves, the easier it is for people to understand how it works and the more they enjoy using it" *[Bibliography, (7)]*. However, when users meet digital elements looking like objects they are used to in real life, they might become confused if they act differently than expected.
Microsoft believes in something which is totally against skeuomorphism. Metro principles do not try to represent real things on a screen, but convey the concept in a way which is appropriate to the digital environment. Therefore, a list of contacts does not look like a real address book, but instead like one designed specifically for a phone or a computer.
These different approaches reflect the controversy about this design principle. For Apple, skeuomorphism improves usability and familiarity; others, like Microsoft, claim that it is unnecessary and ineffective.

3. A TECHNICAL COMPARISON

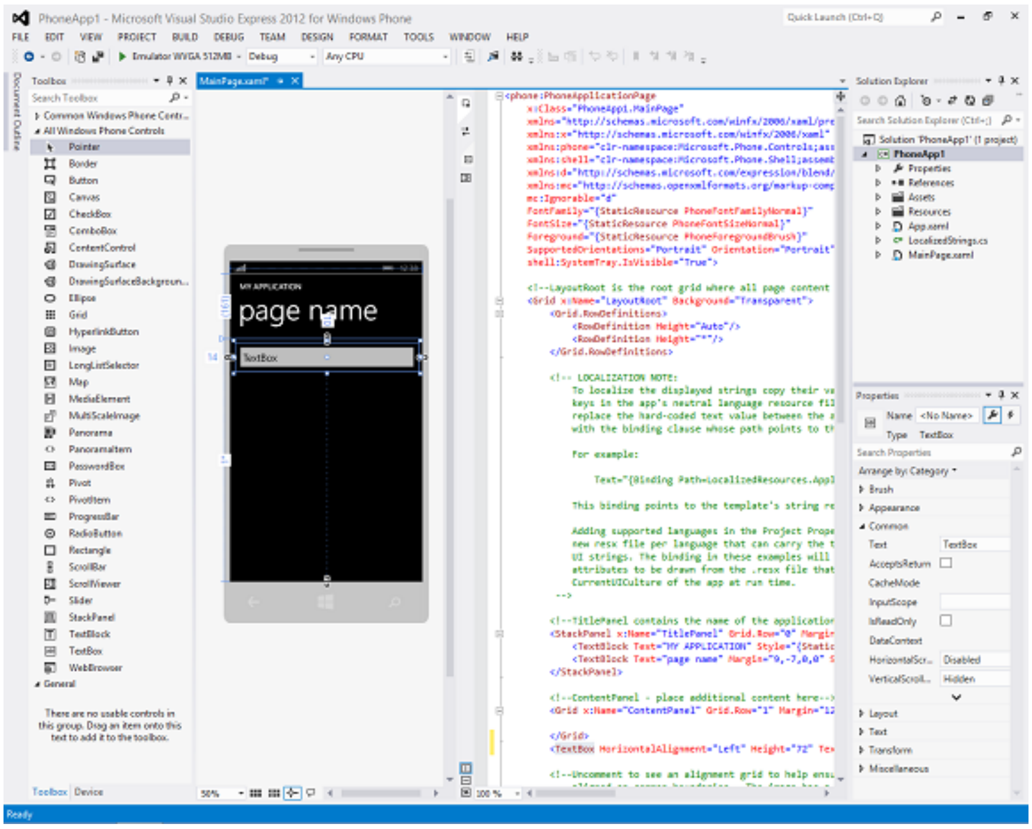
3.1. ADDING CONTROLS AND HANDLING EVENTS

When starting to create an app, the UI must be set up with controls objects such as buttons, text boxes and sliders. The typical process is to add the control, then to set properties such as dimensions or color, and finally to assign actions to the control event handlers. While Windows Phone uses Visual Studio as its integrated development environment (IDE), iOS has Xcode and its visual editor Interface Building. Both IDEs help developers to design their apps, reducing the amount of code needed. For example when a new project is set up, choosing one of the various templates, Interface Building shows the Storyboard file and the main View Controller is instantiated. The same happens with Visual Studio, where a new project comes with the main files already present, like **App.xaml** and **MainPage.xaml.**

3.1.1. Windows Phone

There are three ways for adding and managing a control to an app UI:

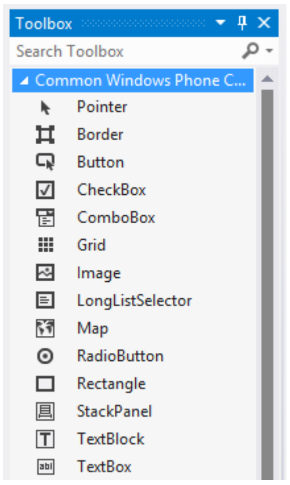
- Using the *Toolbox* in Visual Studio.
- Using the *XAML view*.
- Writing code programmatically.



```
<phone:PhoneApplicationPage
    x:Class="PhoneApp1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pre
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;ass
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assemb
    xmlns:d="http://schemas.microsoft.com/expression/blend/
    xmlns:mc="http://schemas.openxmlformats.org/markup-comp
    mc:Ignorable="d"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

    <!--LayoutRoot is the root grid where all page content
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!-- LOCALIZATION NOTE:
            To localize the displayed strings copy their va
            keys in the app's neutral language resource fil
            replace the hard-coded text value between the a
            with the binding clause whose path points to th

            For example:

                Text="{Binding Path=LocalizedResources.Appl

            This binding points to the template's string re

            Adding supported languages in the Project Prope
            new resx file per language that can carry the t
            UI strings. The binding in these examples will
            attributes to be drawn from the .resx file that
            CurrentUICulture of the app at run time.
         -->

        <!--TitlePanel contains the name of the application
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin
            <TextBlock Text="MY APPLICATION" Style="{Static
            <TextBlock Text="page name" Margin="9,-7,0,0" S
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12

        </Grid>
        <TextBox HorizontalAlignment="Left" Height="72" Tex

        <!--Uncomment to see an alignment grid to help ensu
```

Visual Studio 2012

Visual Studio presents many instruments for developing the UI, like the *Toolbox, Design view, XAML view,* and the *Properties window.*
The Visual Studio *Toolbox* shows a list of all the controls that can be added to the UI.



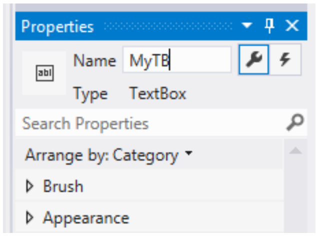
Visual Studio Toolbox

When a control is double-clicked in the *Toolbox*, or dragged to the *Design view*, it is also added to the *XAML view*.
This is an example that shows how to add a `Button` in XAML and in code:

```
XAML
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
   <Button Height="72" Width="160" Content="Click Me"/>
</Grid>
```

```
C#
Button myButton = new Button();
myButton.Width = 160;
myButton.Height = 72;
myButton.Content = "Click Me";
// Attach it to the Grid named ContentPanel as a child
ContentPanel.Children.Add(myButton);
```

Every control is referenced by a unique name, that can be changed in the Visual Studio *Properties window* or in *XAML*. In the *Properties window*, the name can be set in the *Name* text box.



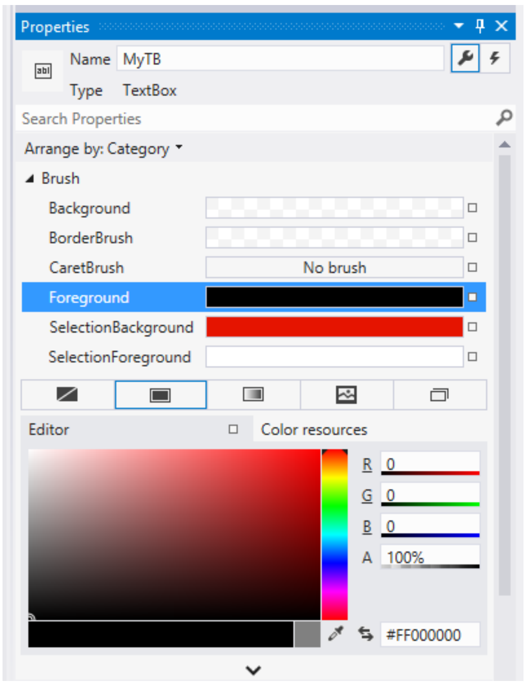
Name MyTB
Type TextBox

Properties window

In the *XAML view*, the *Name* field must be edited.

```
<TextBox x:Name="MyTB" HorizontalAlignment="Left"
```

To change the other control properties, like the appearance or the content, a similar sequence of steps is needed. For example, to set the foreground color for a `TextBox`, the *Properties window* can be used.



Properties
Name MyTB
Type TextBox
Search Properties
Arrange by: Category
Brush
Background
BorderBrush
CaretBrush No brush
Foreground
SelectionBackground
SelectionForeground
Editor
Color resources
R 0
G 0
B 0
A 100%
#FF000000

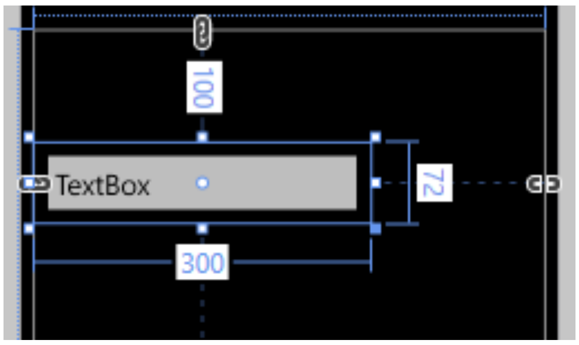
The `Foreground` property can be changed also in the *XAML view*.

```
XAML
<TextBox x:Name="MyTB" HorizontalAlignment="Left" Height="72"
Margin="10,10,0,0" Text="TextBox" VerticalAlignment="Top" Width="460"
```

Finally, it can be set directly in code.

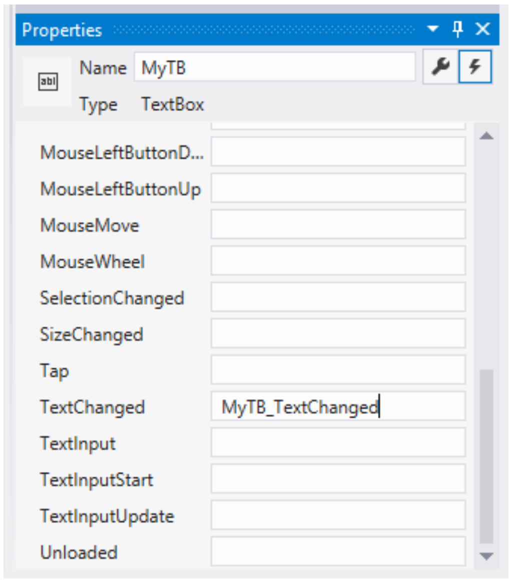
```
C#
SolidColorBrush scb = new SolidColorBrush(Colors.Red);
MyTB.Foreground = scb;
```

Changing properties such as width, height and margin can be done simply by dragging the control in *Design view*.



Design view

User interaction with controls generates events. For example, when the user clicks nearly every control, a `Click` event is raised, and it can be managed with a method called an event handler. An event handler can be created in all the three ways mentioned above.
The *Events tab* of the *Properties window* lists all the events available for the selected control. To create an event handler for a specific user action, the event name must be set.



MouseLeftButtonD...
MouseLeftButtonUp
MouseMove
MouseWheel
SelectionChanged
SizeChanged
Tap
TextChanged MyTB_TextChanged
TextInput
TextInputStart
TextInputUpdate
Unloaded

This will create the event handler in the code editor, and the developer can then choose there what to do. The following code edits the *Text* property of a `TextBlock` named MyBlock to "You entered text!" when text in the `TextBox` MyTB is changed. In fact this change raises the `TextChanged` event, handled by the `MyTB_TextChanged` method.

```
C#
private void MyTB_TextChanged(object sender, TextChangedEventArgs e)
{   MyBlock.Text = "You entered text!";    }
```

To create an event handler in XAML, the event name and the event handler have to be specified. The following XAML shows the same event and event handler seen before.

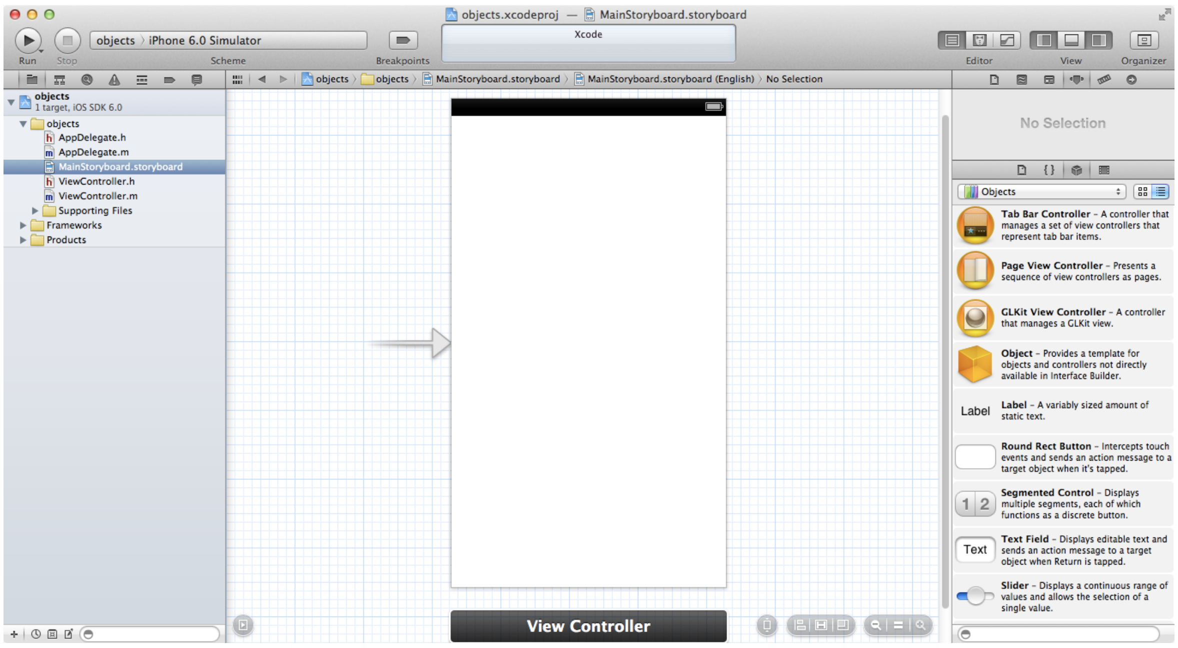
```
XAML
<TextBox x:Name="MyTB" TextChanged="MyTB_TextChanged"
HorizontalAlignment="Left" Height="72" Margin="10,10,0,0" Text="TextBox"
VerticalAlignment="Top" Width="460" Foreground="Red"/>
```

The third way to do that is by code, as shown in the following line.

```
C#
MyTB.TextChanged +=new TextChangedEventHandler(MyTB_TextChanged);
```

3.1.2. iOS

Adding and managing a control can be done in a declarative way, or in a programmatic one. Interface Builder has an *Object Library* toolbar, that lists all the available controllers, like the `Table View` and `Tab Bar` ones, controls, such as labels and text fields, and other objects.



Tab Bar Controller – A controller that manages a set of view controllers that represent tab bar items.

Page View Controller – Presents a sequence of view controllers as pages.

GLKit View Controller – A controller that manages a GLKit view.

Object – Provides a template for objects and controllers not directly available in Interface Builder.

Label – A variably sized amount of static text.

Round Rect Button – Intercepts touch events and sends an action message to a target object when it's tapped.

Segmented Control – Displays multiple segments, each of which functions as a discrete button.

Text Field – Displays editable text and sends an action message to a target object when Return is tapped.

Slider – Displays a continuous range of values and allows the selection of a single value.

Xcode

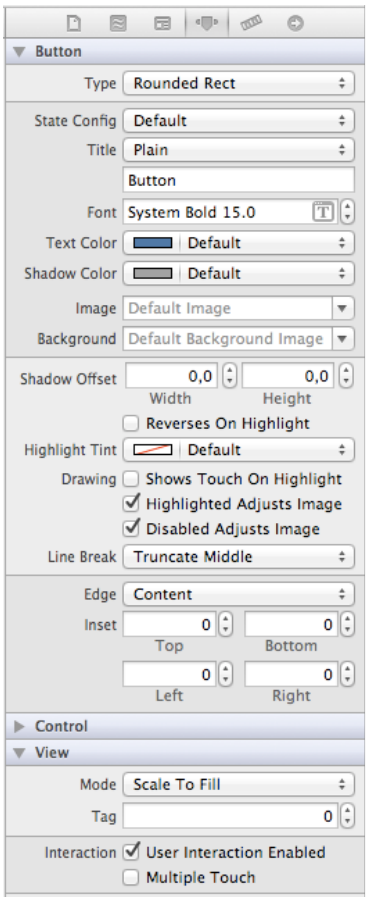
Adding a control, for example a button, to the view can be done by dragging it simply from this list, or by code.

```
UIButton *btn = [UIButton buttonWithType:UIButtonTypeRoundedRect];
//The parameters of CGRectMake are x-y coordinates, width and height.
  btn.frame = CGRectMake(117,252,86,44);
//Add the button to the view.
  [self.view addSubview:btn];
```

Every control has a list of properties, and editing them in code requires specific methods. For example, to set a title to the previously created button the following line of code must be added.

```
[btn setTitle:@"Click me" forState:UIControlStateNormal];
```

The list of properties can be also edited easily in the *Attributes Inspector*.



Button

Type: Rounded Rect

State Config: Default
Title: Plain
Button
Font: System Bold 15.0
Text Color: Default
Shadow Color: Default
Image: Default Image
Background: Default Background Image

Shadow Offset: 0,0 Width, 0,0 Height
☐ Reverses On Highlight
Highlight Tint: Default
Drawing: ☐ Shows Touch On Highlight
✓ Highlighted Adjusts Image
✓ Disabled Adjusts Image
Line Break: Truncate Middle

Edge: Content
Inset: 0 Top, 0 Bottom, 0 Left, 0 Right

Control

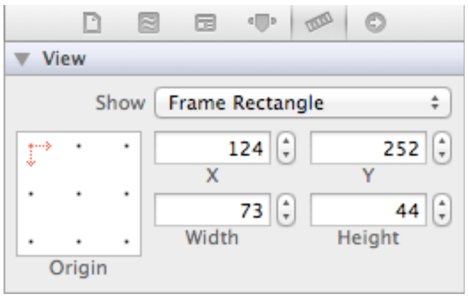
View

Mode: Scale To Fill
Tag: 0

Interaction: ✓ User Interaction Enabled
☐ Multiple Touch

Attributes Inspector

Changing properties such as width, height and margins can be done simply by dragging the control in the view, by creating a new Frame in code, or by editing them in the *Size Inspector*.



View

Show: Frame Rectangle
124 X, 252 Y
73 Width, 44 Height
Origin

Size Inspector

IOS development follows some key methodologies, called design patterns. The Target-Action one defines how the view and controls created in Interface Builder interact with the code written in the view controllers classes. There are two main keywords: `IBAction` and `IBOutlet`. An `IBAction` represents a method in the view controller that is called when the user interacts with a specific object of the view. An `IBOutlet` represents a handler to a view object that allows modifications to the control properties in the view controller.

In the following example, when the user taps the button a "change the label text" message (the action) is sent to the view controller (the target). In the view controller the `IBAction` method `buttonPressed` edits the title property of the label, through an `IBOutlet`. This application function can be implemented in the following steps.

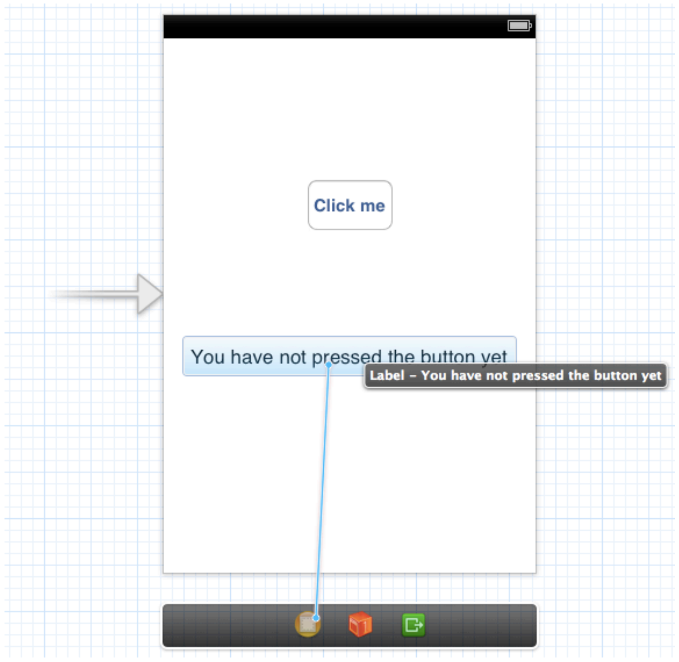
1. In a new project created with the *Single View Application* template, add a button and a label.
2. Set the titles of the button and the label to "Click me" and "You have not pressed the button yet" respectively.
3. In the **ViewController.h** file declare the `IBOutlet` and the `IBAction`.

```
#import <UIKit/UIKit.h>

@interface ViewController: UIViewController

@property (strong, nonatomic) IBOutlet UILabel *label;
-(IBAction)buttonPressed:(id)sender;

@end
```

4. In the **ViewController.m** file implement the `buttonPressed` method.

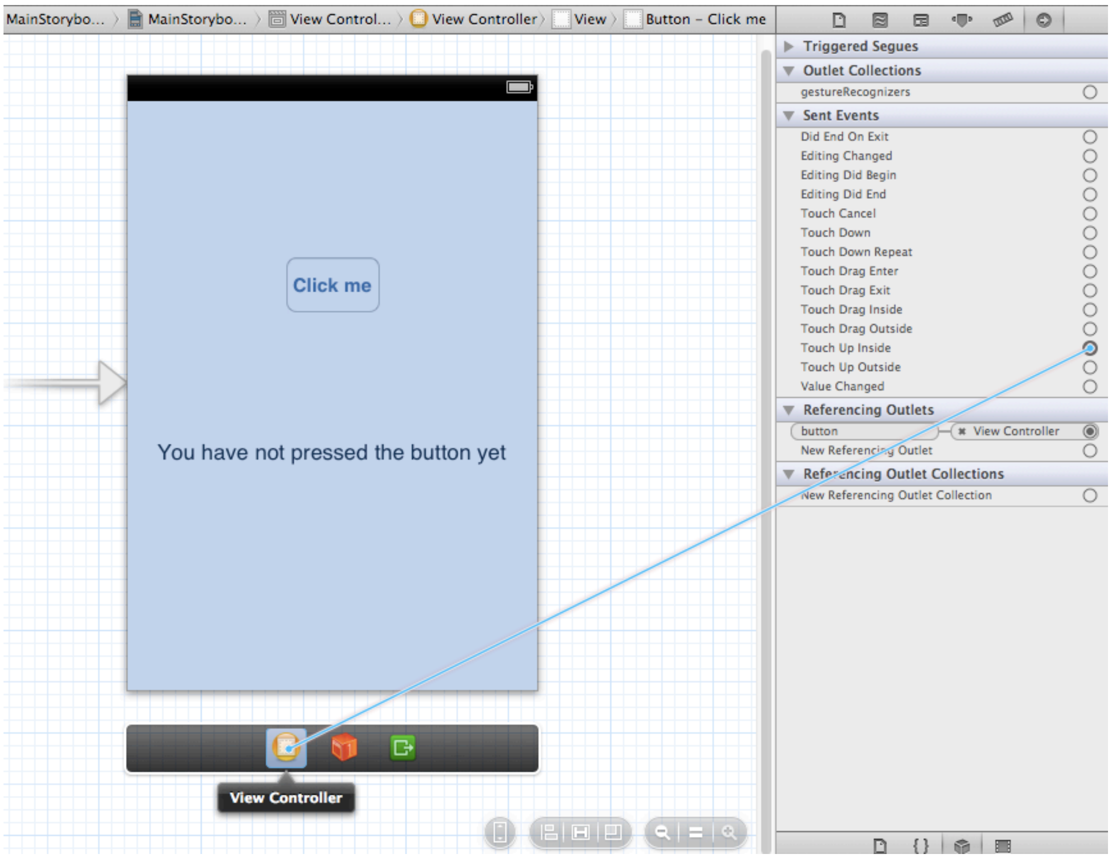
```
#import "ViewController.h"

@interface ViewController ()
@end

@implementation ViewController

-(void)buttonPressed:(id)sender
{
    _label.text = @"You have pressed the button";
}

- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}

@end
```

5. Return to the **MainStoryboard.storyboard** file, hold down the Control key on the keyboard and draw a line from the *View Controller* icon to the label object.


Click me
You have not pressed the button yet
Label – You have not pressed the button yet

Connecting the IBOutlet to the Label control

6. In the black list of the available `IBOutlet`, choose "label".
7. Select the button and display all the events that can be triggered in the *Connections Inspector*. Draw a line from the `Touch Up Inside` event to the *View Controller* icon, and choose the `buttonPressed` method.



Connecting the event to the IBAction

8. Now the connections are established, and when the user click the button, the label changes its text.

Connecting an action to a control can be done also programmatically. In the previous example, it is required to add an `IBOutlet` for the button, and to write the following code in the `viewDidLoad` method of the **ViewController.m** file.

```
- (void)viewDidLoad
{
   [super viewDidLoad];
   // "button" is the name of the IBOutlet declared in ViewController.h
   [_button addTarget:self
               action:@selector(buttonPressed:)
     forControlEvents:UIControlEventTouchUpInside];
}
```

3.2. SPACE MANAGEMENT

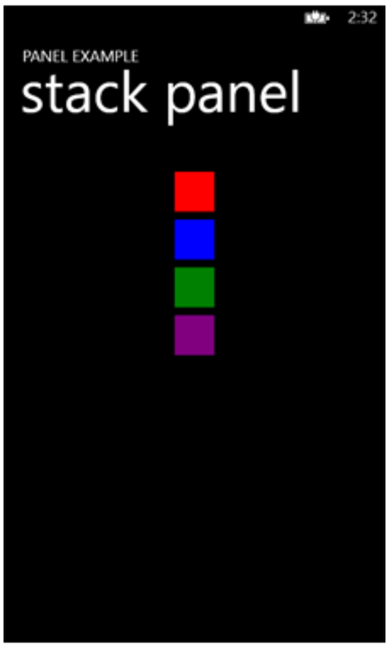
The main asset of current smartphones is undoubtedly their screen. It represents the primary interaction medium between the user and the device, and therefore particular care must be paid on how content looks on it.
The term "layout" refers to the process of positioning and resizing objects on the app interface, which is fundamental to give the app a polished and ordinate view. Both Windows Phone and iOS try to help developers with simple yet powerful tools, called respectively "*dynamic*" and "*Auto*" layout. They are particularly important for managing different screen sizes and orientation changing.

3.2.1. Windows Phone

LAYOUT

Windows Phone provides two types of layout, an absolute and a dynamic one.
In the *absolute* layout, the x/y coordinates of the controls are explicitly set. The container supporting this layout is the `Canvas`, and objects are positioned by using the `Canvas.Left` and `Canvas.Top` properties. They represent respectively the distance from the left and the top margin of the `Canvas` area. This layout is not usually recommended because it does not adapt to different screen resolutions, or orientation changes.
The *dynamic* layout, instead, takes care of these problems, positioning controls in relation with their parents, and with one another. Therefore, objects rearrange their position and dimensions automatically when a change occurs.
Instead of explicit dimensions, it is recommended to set the `Height` and `Width` properties to *auto*, or to *. These are special values for automatic and proportional sizing. *Auto* sizing allows controls to fit their content, even if it changes. *Star* sizing distributes the space available using weighted proportions. For example, if the values * and 3* are assigned to two rows of a grid layout in the height property, the second row will be three times taller than the first one.
Another suggestion is to set the `MinWidth` and `MinHeight` properties for controls containing text, so that the text is always readable.
The two containers supporting dynamic layout are the `StackPanel` and the `Grid`.
In a `StackPanel` layout, the child elements are arranged into a single line, that can be oriented horizontally or vertically. By default the `Orientation` property is vertical.
This is a simple example of how the `StackPanel` works:

```xml
XAML
<StackPanel Margin="20">
  <Rectangle Fill="Red" Width="50" Height="50" Margin="5"/>
  <Rectangle Fill="Blue" Width="50" Height="50" Margin="5"/>
  <Rectangle Fill="Green" Width="50" Height="50" Margin="5"/>
  <Rectangle Fill="Purple" Width="50" Height="50" Margin="5"/>
</StackPanel>
```



Example of a StackPanel

The `Grid` is the default layout panel, and it is the most flexible one. As its name suggests, it organizes the area in rows and columns. Their number and size can be specified, and controls can be placed in the resulting cells.

The following is an example of a grid with three rows and two columns:

```
XAML
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="auto"/>
    <RowDefinition Height="auto"/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="2*"/>
    <ColumnDefinition Width="*"/>
  </Grid.ColumnDefinitions>
</Grid>
```

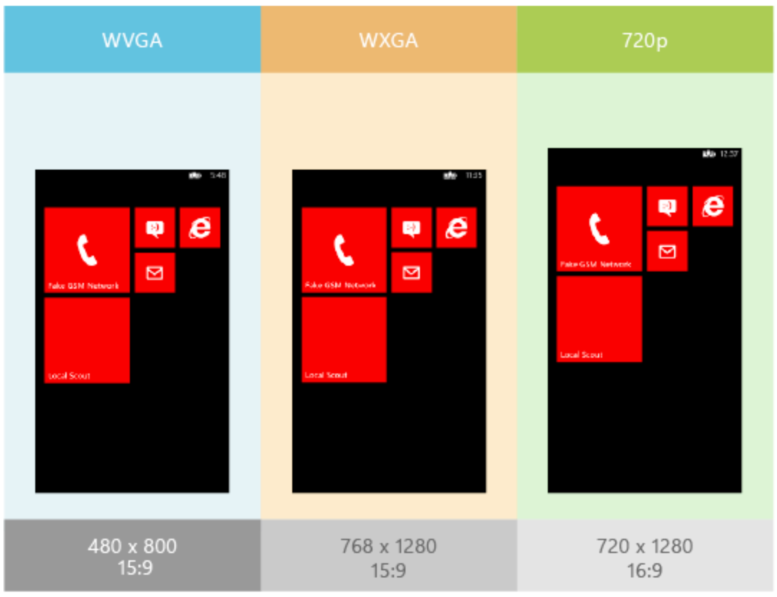
The first two rows will size to fit their content, while the third one will fill up the available space. The second column is two times the first one.

To place a control in a specific cell, its position has to be specified:

```
XAML
<Grid>
  <!--Row and column definitions-->
  <Button Content="1st row 2nd column" Grid.Row="0" Grid.Column="1"/>
</Grid>
```

SCREEN SIZES

While Windows Phone 7.x supported only the WVGA resolution (480 x 800 pixels), Windows Phone 8 adds also WXGA (768 x 1280) and 720p (720 x 1280). The two first resolutions have the same aspect ratio, 15:9, while the aspect ratio of the third is 16:9.



Windows Phone 8 supported resolutions

This means that with the new OS the developer has to take care of adapting apps to different resolutions. In fact, what seems perfectly disposed with a particular aspect ratio, may appear completely different with the other one.
The best solution is to use a dynamic layout instead of an absolute one, deleting margins and coordinates that change on different resolutions, and positioning controls with the auto and star sizing. This lets objects stretch and fit automatically on the screen while maintaining their relative positions. It is important that controls do not shrink under 8 mm, because they will become hard to press for users: to avoid this, developers can use the `MinHeight` and `MinWidth` properties seen before.
However, in some cases it is necessary to load different images (like backgrounds) for each resolution. The following example shows how to detect the screen size and then load the correct file at run time:

1. In the project file, add the images for WVGA, WXGA and 720p resolutions. In this example, the files are `MyImage.screen-wvga.png`, `MyImage.screen-wxga.png`, and `MyImage.screen-720p.png`.
2. Set the Copy to Output Directory property of the images to *copy always*.
3. Add a class named **ResolutionHelper.cs** to the project, and then write the following code into the new class.

```
C#
public enum Resolutions { WVGA, WXGA, HD720p };

public static class ResolutionHelper
{
  private static bool IsWvga
  {  get { return App.Current.Host.Content.ScaleFactor == 100; }  }
  private static bool IsWxga
  {  get { return App.Current.Host.Content.ScaleFactor == 160; }  }
  private static bool Is720p
  {  get { return App.Current.Host.Content.ScaleFactor == 150; }  }
  public static Resolutions CurrentResolution
  {
    get
    {
      if (IsWvga) return Resolutions.WVGA;
      else if (IsWxga) return Resolutions.WXGA;
      else if (Is720p) return Resolutions.HD720p;
      else throw new InvalidOperationException("Unknown resolution");
    }
  }
}
```

4. Add a class named **MultiResImageChooser.cs** that contains the following code. This class uses the **ResolutionHelper.cs** class already created to determine the resolution of the device. Then, it returns a new BitmapImage created from the URI of the image that corresponds to the detected resolution.

```
using System.Windows.Media.Imaging;

public class MultiResImageChooserUri
{
  public Uri BestResolutionImage
  {
    get
    {
      switch (ResolutionHelper.CurrentResolution)
      {
        case Resolutions.HD720p:
          return new Uri("Assets/MyImage.screen-720p.jpg", 
UriKind.Relative);
        case Resolutions.WXGA:
          return new Uri("Assets/MyImage.screen-wxga.jpg", 
UriKind.Relative);
        case Resolutions.WVGA:
          return new Uri("Assets/MyImage.screen-wvga.jpg", 
UriKind.Relative);
        default:
          throw new InvalidOperationException("Unknown resolution");
      }
    }
  }
}
```

5. In **MainPage.xaml**, add an Image element and bind its `Source` property to the URI returned by the **MultiResImageChooser.cs** class.

```
XAML
<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
  <Image Source="{Binding BestResolutionImage,
                Source={StaticResource MultiResImageChooser}}"/>
</Grid>
```

6. In the <*Application*> element of **App.xaml**, add the following **xmlns** namespace mapping.

```
XAML
xmlns:h="clr-namespace:MultiResSnippet"
```

7. In **App.xaml**, add the following application resource.

```
XAML
<!--Application Resources-->
<Application.Resources>
   <h:MultiResImageChooser x:Key="MultiResImageChooser"/>
</Application.Resources>
```

For splash screens it is advisable to use a single image file named `SplashScreenImage.jpg` that is 768 × 1280 pixels in size (WXGA). It is the best solution since it has the same aspect ratio as WVGA with higher quality, and in 720p resolution it only stretches a bit. If more precision is needed, the developer must add images with the following file names in the root folder of the app project:

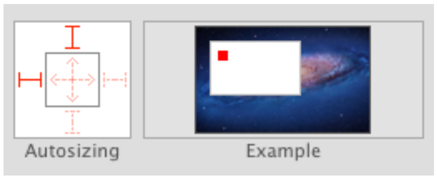
- SplashScreenImage.Screen-WVGA.jpg
- SplashScreenImage.Screen-WXGA.jpg
- SplashScreenImage.Screen-720p.jpg

The WXGA resolution is suggested also for Tiles and Assets such as graphics, video, audio and icons.

3.2.2. iOS

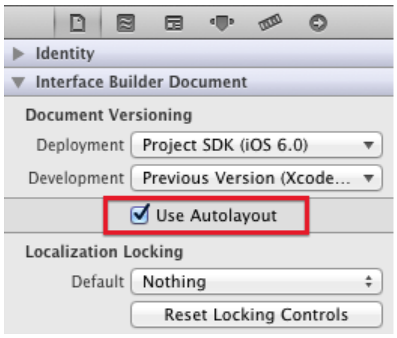
LAYOUT

With iOS 6, Apple has made an important change in the way developers design the user interface of their apps, specifically regarding layout. The new feature is called *Auto Layout*, and replaces the old system of *Auto Sizing*, informally known as "springs and struts" model. This deprecated method consisted of rules that relate controls to their superviews. "Springs" stand for the height and width of the object, that can grow or lower depending on the superview. "Struts" stand for the fixed or flexible margins, that determine the distance from the edges.



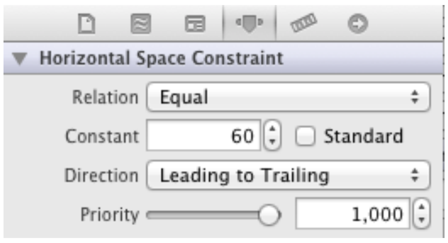
The old Auto Sizing system

However, Auto Sizing had strong limits, and a considerable amount of coding was often necessary for complex layouts.
Auto Layout, instead, creates relationships not only between the elements and the view, but also among the objects themselves. This is permitted by the use of constraints, that are real objects of the `NSLayoutConstraint` class, and therefore have attributes that can be changed. A costraint specifies a geometric relationship between two objects, for example declaring that they must have the same width, or that there must be a 20-point horizontal padding between them. Auto Layout considers all the constraints that have been set and calculates the best layout that follows all of them. This is called "designing by intent", because the developer describes only what they want, in a descriptive way, without the need to specify how it should be accomplished, by explicit coordinates.
Auto Layout is primarily aimed at managing different screen sizes and orientation changes correctly, but it is helpful also for internationalization. In fact, it has always been a hard work to deal with other languages, some of them having long words, and labels and buttons have to fit them correctly. With iOS 6, it is much easier, because many controls have an "intrinsic content size". This means that they calculate their size automatically, based on their content. For example, a label arranges its width in relation with the text it contains.
By default, Auto Layout is already active, and to turn it off it must be unchecked.



Identity
Interface Builder Document
Document Versioning
Deployment: Project SDK (iOS 6.0)
Development: Previous Version (Xcode...
☑ Use Autolayout
Localization Locking
Default: Nothing
Reset Locking Controls

Although constraints can be created by code, the recommended approach is to use Interface Building. Simply dragging an object on the view makes constraints visible as blue I-beam, that connect it to the edges or to other objects. Then, constraints can be edited by selecting them, and changing their properties in the *Size Inspector*.

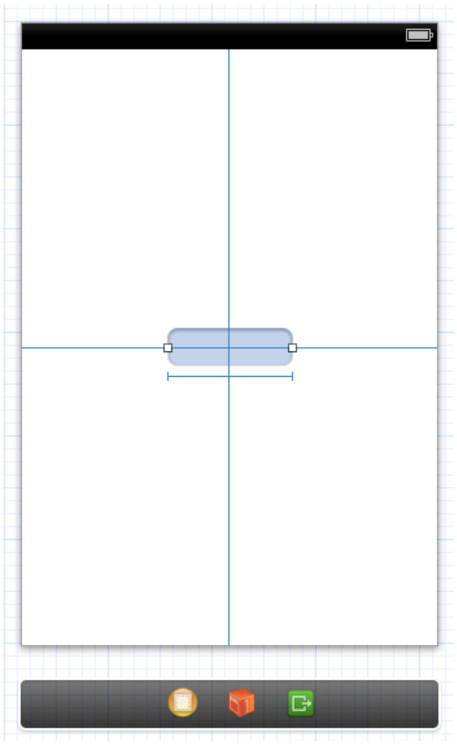
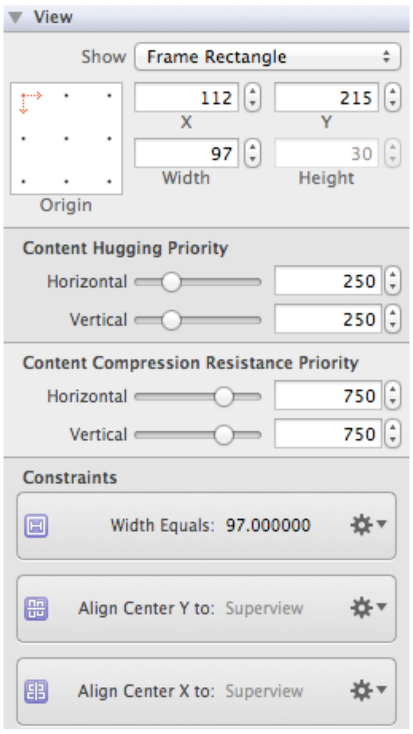

Horizontal Space Constraint

Relation: Equal
Constant: 60 ☐ Standard
Direction: Leading to Trailing
Priority: 1,000

Size Inspector

The following example shows how to deal with constraints, and the usefulness of the intrinsic content size. A control is positioned in the center of the view, and another one on top of the screen, aligning the left edges. If the "*Size to fit content*" costraint is set, when the first control expands the second one will follow it in a dynamic way.

1. Create a new project in Xcode based on the *Single View Application* template.
2. Position a `Text Field` in the center of the view, following the guide lines. This creates three constraints, visible in the *Size inspector*.



View

Show: Frame Rectangle
X: 112 Y: 215
Width: 97 Height: 30
Origin

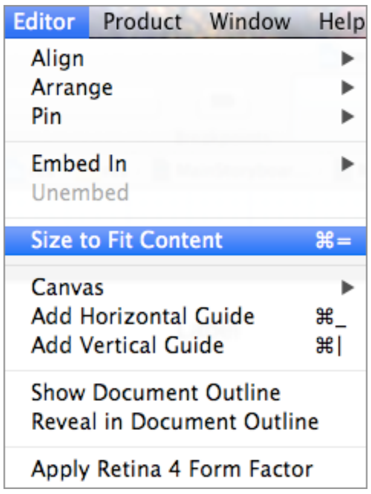
Content Hugging Priority
Horizontal: 250
Vertical: 250

Content Compression Resistance Priority
Horizontal: 750
Vertical: 750

Constraints
Width Equals: 97.000000
Align Center Y to: Superview
Align Center X to: Superview

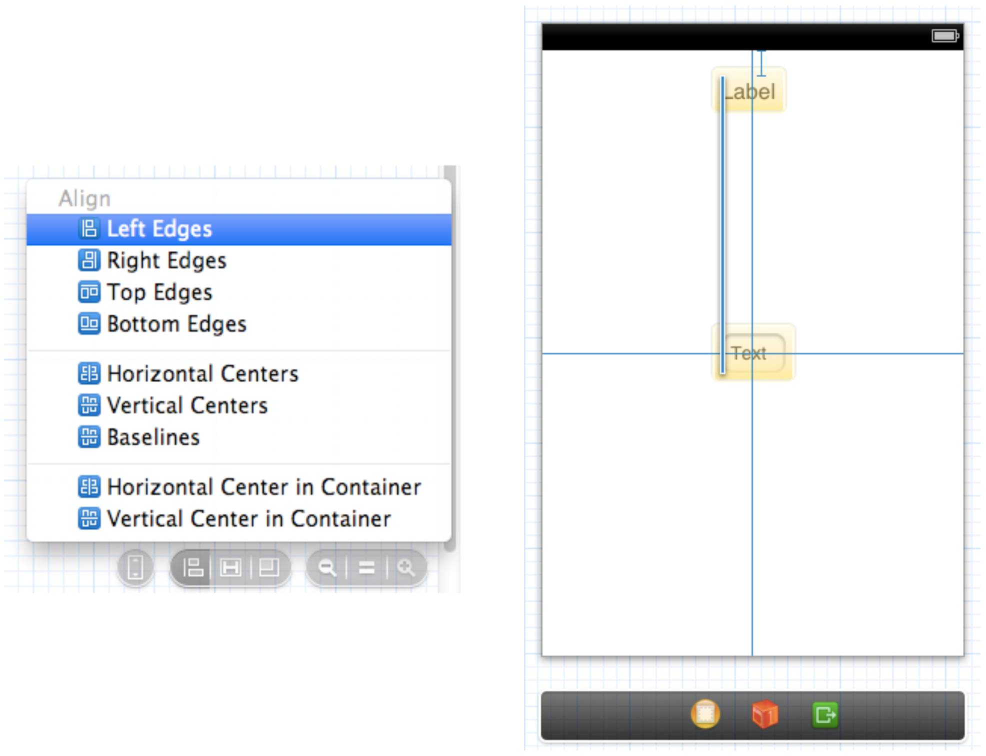
The visual editor with the guide lines, and the created constraints in the Size Inspector

3. Write something in the `Text Field`, and select "*Size to Fit Content*" on the *Editor* menu, deleting the *Width* constraint.



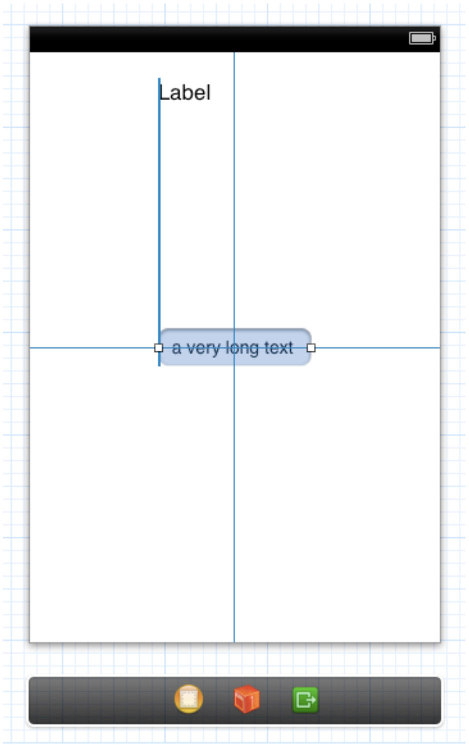
The Editor menu, with the Size to Fit Content constraint

4. Put a `Label` at the top of the view, and select both the `Text Field` and the new control.
5. Select "*Align*" and "*Left edges*" on the *Editor* menu, or simply click the *Align* button at the bottom of the canvas area.



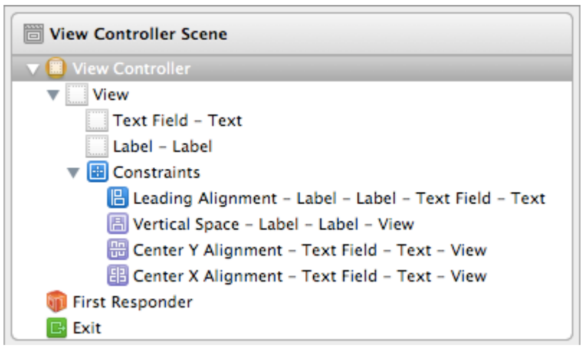
Aligning the left edges of the Label and the Text Field

6. Now, if a longer text is written in the `Text Field`, the "*Size to fit content*" costraint adapts the control width to show the entire text, and the `Label` will move to maintain the "*Aligning left edges*" costraint.



The Label moves left to maintain the Aligning constraint

7. All the constraints are visible also on the *Document Outline* panel.



View Controller Scene
View Controller
View
Text Field – Text
Label – Label
Constraints
Leading Alignment – Label – Label – Text Field – Text
Vertical Space – Label – Label – View
Center Y Alignment – Text Field – Text – View
Center X Alignment – Text Field – Text – View
First Responder
Exit

Document Outline panel

SCREEN SIZES

Until the introduction of the iPhone 5, all its predecessors shared a 3.5-inch touchscreen, with a 3:2 aspect ratio; the only difference was the resolution: since the introduction of the Retina display with the iPhone 4, it is 640 x 960 pixels, exactly two times the old one. The new iPhone 5 has a 4-inch screen, with an aspect ratio of 16:9 and a 1136 × 640 resolution.

Dealing with different resolutions is simplified by Apple's distinction between pixels and points. Excluding the latest iPhone, the screens of the other models can be divided into standard-resolution, and high-resolution. Using points instead of pixels makes development almost device-independent, letting developers specify the position and size of controls in a relative way. The conversion from points to pixels is then handled by the system frameworks, that use a scale factor of 1.0 for standard-resolution and one of 2.0 for the high-resolution.

The `UIKit` framework, for example, takes care of rendering text and standard controls, like buttons, sliders, table views and so on, correctly. The developer is only required to provide two images, one for each resolution, and `UIKit` automatically chooses the correct one.

The naming convention states the following rules:

- Standard: *<ImageName>.<filename_extension>*
- High-resolution: *<ImageName>@2x.<filename_extension>*

It is very important that the *<ImageName>* is the same, because it is needed to refer to an image in code. For example, with the following code the `UIImage` class will search for the high-resolution file if it is supported by the device.

```
Objective-C
UIImage *image = [UIImage imageNamed:@"button.png"];
```

There is a method to detect the screen size, in points:

```
[[UIScreen mainScreen] bounds].size.height
```

For an iPhone 5 the result is 568, while for all the other models it is 480.

This method is useful for loading different images with different resolutions, but generally Auto Layout does the hard work. In fact to support the iPhone 5, it is enough to add a launch image called "`Default-568h@2x.png`", with a resolution of 1136 x 640. Then, if Auto Sizing or Auto Layout are set correctly, the app will work well on all resolutions.

3.3. ORIENTATION MANAGEMENT

When the user turns to device, the accelerometer detects the change and the system sends a notification to the app. If the notification is correctly handled, the app will turn its interface together with the screen. There are different techniques to rearrange the content when a change occurs, and it is important to find the right one for the best result, depending on the case. A great help is given by the automatic layouts (dynamic and Auto) that both OSes have. The next paragraphs explain these different techniques and when to use them.

3.3.1. Windows Phone

Windows Phone supports three screen orientations: *Portrait*, *Landscape Left* and *Landscape Right*. The default orientation is *Portrait*, and to add landscape support the `SupportedOrientations` property must be set to `PortraitOrLandscape` in the **MainPage.xaml**. There is no option to specify left-only or right-only landscape orientation, so both must be implemented.
In these two landscape orientations, the *Status Bar* and the *Application Bar* remain on the side of the screen that has the Power and the Start button, respectively. Thus, the *Status Bar* changes its position on the left of the screen when the orientation is *Landscape Left*, and on the right when it is *Landscape Right*.
To manage all orientations it is necessary to use a dynamic layout. This lets the app adapt its content and guarantees the best user interaction. Two suggested techniques are the scrolling technique and the grid layout one.

The scrolling technique is used when the app content does not fit on the screen when the orientation switches to landscape. It consists in a `StackPanel` control that is placed within a `ScrollViewer` control. The `StackPanel` orders the elements one after another and the `ScrollViewer` control enables scrolling through the `StackPanel`.

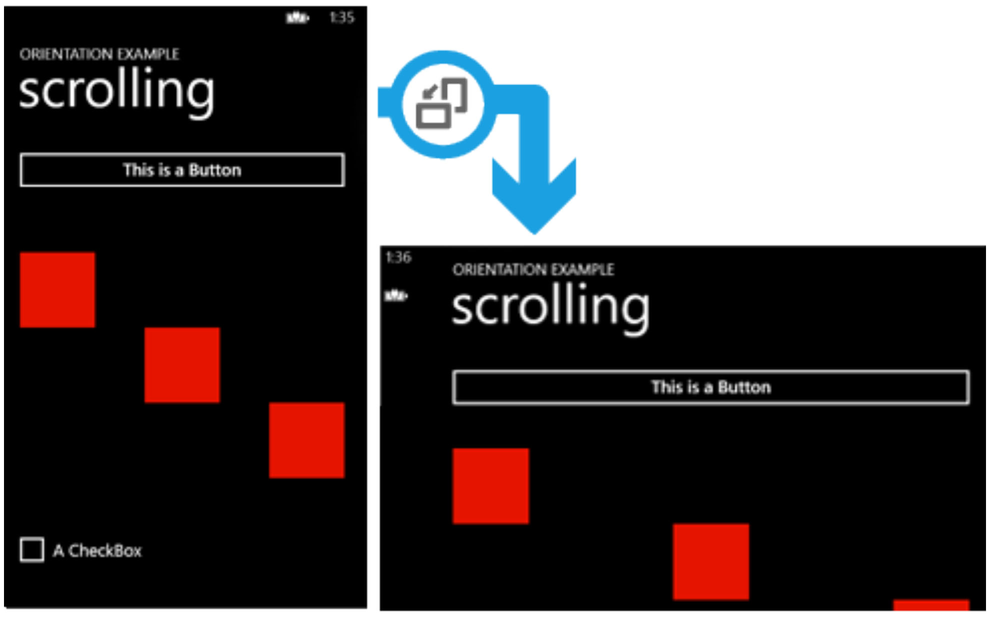
To use the scrolling technique, some important steps must be followed:

- Change the `SupportedOrientations` property of the page to `PortraitOrLandscape`.
- Replace the default `Grid` in the `Content Panel` section with a `ScrollViewer` and a `StackPanel`.

The following example shows how to use this technique.

```
XAML
<ScrollViewer x:Name="ContentGrid" Grid.Row="1"
VerticalScrollBarVisibility="Auto">
  <!--You must apply a background to the StackPanel control or you will
be unable to pan the contents.-->
  <StackPanel Background="Transparent">
    <Button Content="This is a Button"/>
    <Rectangle Width="100" Height="100" Margin="12,0"
HorizontalAlignment="Left" Fill="{StaticResource PhoneAccentBrush}"/>
    <Rectangle Width="100" Height="100" HorizontalAlignment="Center"
    Fill="{StaticResource PhoneAccentBrush}"/>
    <Rectangle Width="100" Height="100" Margin="12,0"
HorizontalAlignment="Right" Fill="{StaticResource PhoneAccentBrush}"/>
    <CheckBox Content="A CheckBox"/>
  </StackPanel>
</ScrollViewer>
```

The output shows that the portrait orientation does not require scrolling, while the landscape one does.



The scrolling techinque

The grid layout technique, as its name suggests, uses a `Grid` to position UI elements. When the orientation changes, controls are programmatically repositioned into different cells of the `Grid`.

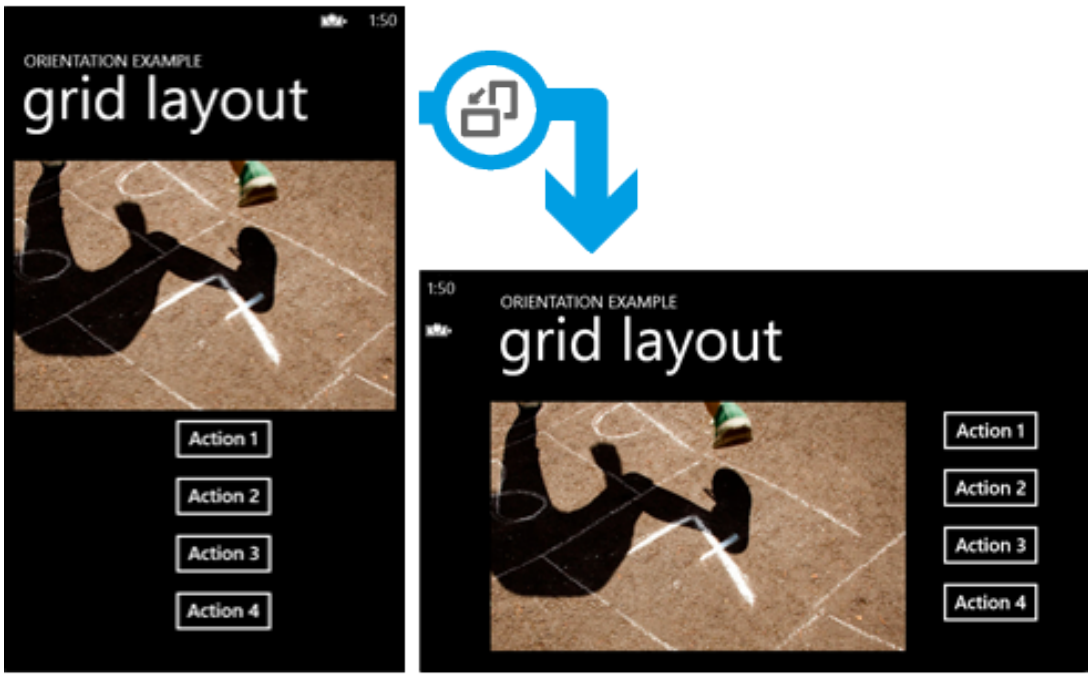
Using the grid layout technique requires the following steps:

- Change the `SupportedOrientations` property of the page to `PortraitOrLandscape`.
- Use a `Grid` for the content panel (as set by default).
- Create an `OrientationChanged` event handler and add code to reposition elements in the `Grid`.

The following example creates a 2 x 2 `Grid` to position an image and a collection of buttons.

```
XAML
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*"/>
  </Grid.ColumnDefinitions>
  <Image x:Name="Image" Grid.Row="0" Grid.Column="0" Stretch="Fill"
Source="img.jpg" HorizontalAlignment="Center" Height="300" Width="500"/>
  <StackPanel x:Name="buttonList" Grid.Row="1" Grid.Column="0"
HorizontalAlignment="Center">
    <Button Content="Action1"/>
    <Button Content="Action2"/>
    <Button Content="Action3"/>
    <Button Content="Action4"/>
  </StackPanel>
</Grid>
```

The output shows how the position of the buttons changes according to the orientation.



The grid layout techinque

3.3.2. iOS

IOS can support four orientations: portrait, upside down, landscape left and landscape right. By default, the upside down orientation is not selected, and its use is deprecated.
When an orientation change occurs, the system sends a `UIDeviceOrientationDidChangeNotification` that `UIKit` framework collects, and the interface is changed automatically if the new orientation is supported.
With iOS 6 autorotation techniques are changed, and there are two new important methods: "`shouldAutorotate`" and "`supportedInterfaceOrientations`". The following example shows how to use them.

```
- (BOOL) shouldAutorotate
{
   //Returns whether the view controller’s contents should auto rotate
   return YES;
}

- (NSUInteger)supportedInterfaceOrientations
{
   //Returns all of the interface orientations that the view controller
supports.
   return UIInterfaceOrientationMaskAllButUpsideDown;
}
```

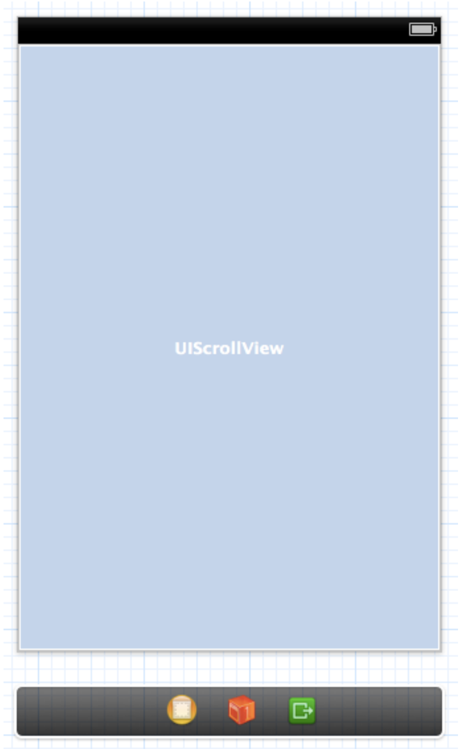
If the first method returns "NO", the second one is not even called, and no rotation occurs.

Auto Layout is a powerful instrument for managing orientation changes. Positioning controls in a relative way lets automatic rearrangement takes place when switching to a new orientation. With the right constraints, layout can adapt to the device rotation correctly, stretching or shrinking objects if needed.
There are basically two suggested techniques to avoid hard-coded repositioning of elements, when Auto Layout is not sufficient: using a `ScrollView`, or creating an alternative interface.

The `ScrollView` technique is used when the content does not fit in the `Landscape` mode. To avoid this problem, the `ScrollView` lets the user scroll vertically the page and see all the content.

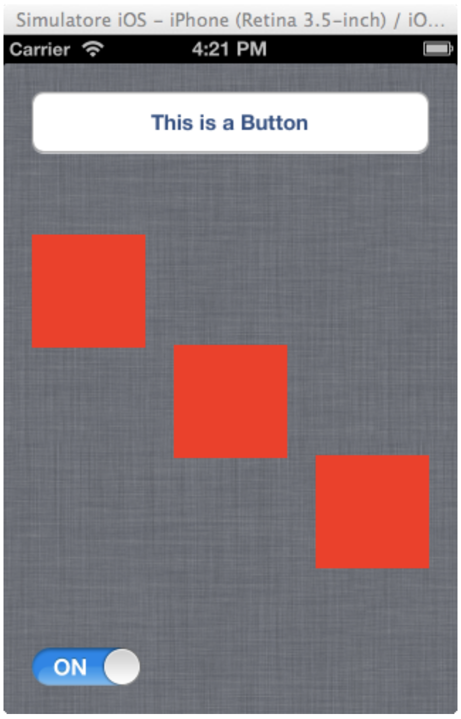
The following example shows how to use this technique.

1. In Interface Builder, add a `ScrollView` over the standard `View`.



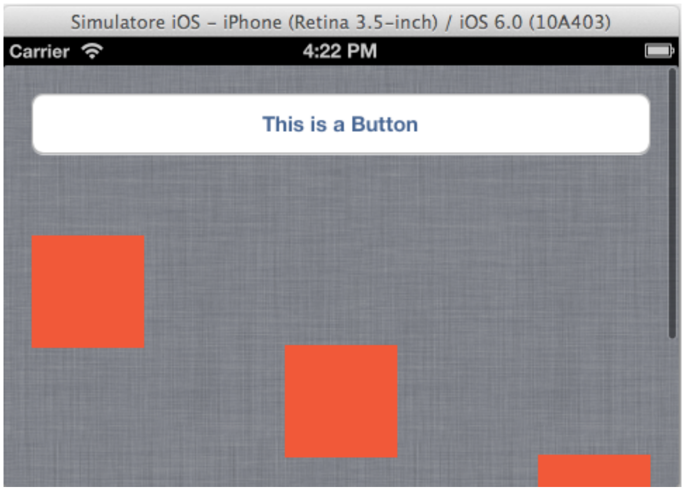
The ScrollView in the visual editor

2. Starting from the top, add a `Button`, three `ImageViews`, and a `Switch`. Set the `Background` of the `ScrollView` to the Scroll View Textured color, and those of the `ImageViews` to red.



The view with the various controls

3. Without any more work, Auto Layout arranges the layout in the `Landscape` mode properly. Simply scroll the view to see all the content.

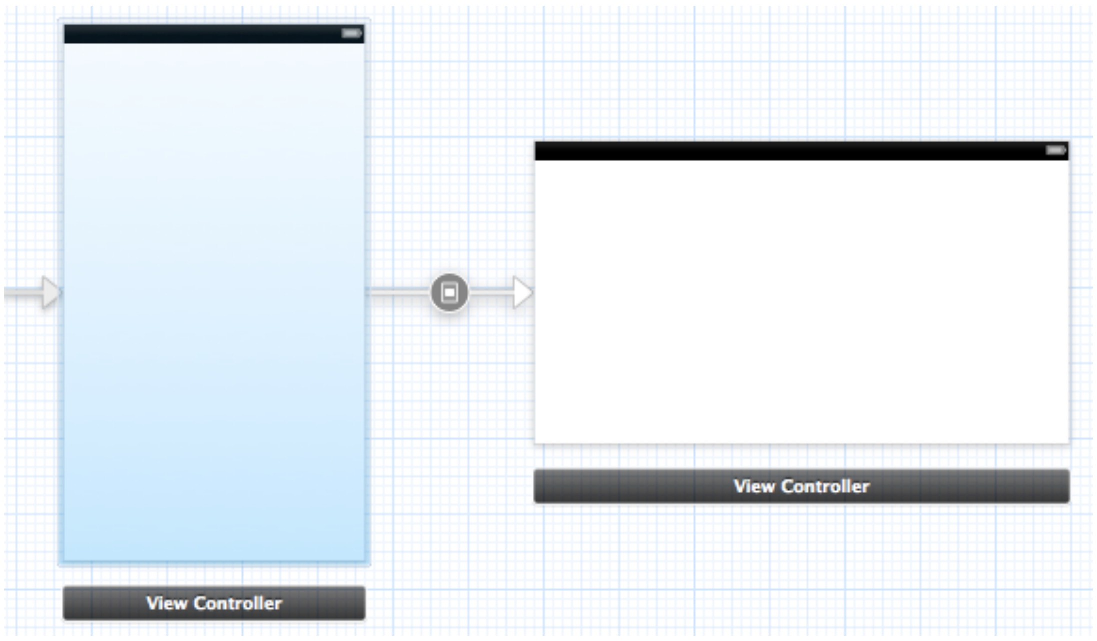

Simulatore iOS - iPhone (Retina 3.5-inch) / iOS 6.0 (10A403)
Carrier
4:22 PM
This is a Button

The view in the Landscape mode

The second technique is used when the layout of the landscape mode is quite different from the portrait one. Another method is to modify the position of controls programmatically, but it requires a lot of hard-coding and it is suggested only for little changes to the layout.
To create an alternative interface two view controllers are needed, one for a portrait-only orientation, and the other for a landscape-only orientation. The primary view controller must be registered for the `UIDeviceOrientationDidChangeNotification`, and takes care of presenting or dismissing the other view controller when an orientation change occurs. To achieve this a particular connection is used, called "*Segue*", which represents a transition from one view to another.

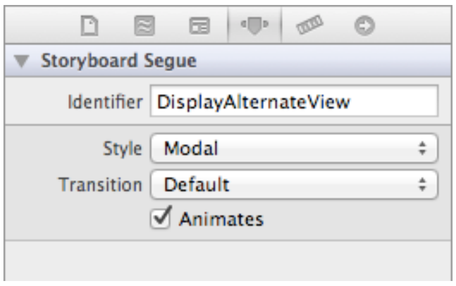
The following example shows how to use this technique.

1. Create a new project in Xcode based on the *Single View Application* template.
2. Add another `View Controller`. Select the `Landscape` orientation in the *Attributes Inspector* for the new `View Controller`, and the `Portrait` orientation for the first one.
3. Select the primary `View Controller`, press "ctrl" and drag a line to the second `View Controller`. This will create a *Segue* connection.



The Segue connection

4. Set the *Segue Identifier* as "*DisplayAlternateView*".



Identifier DisplayAlternateView

Style Modal

Transition Default

✓ Animates

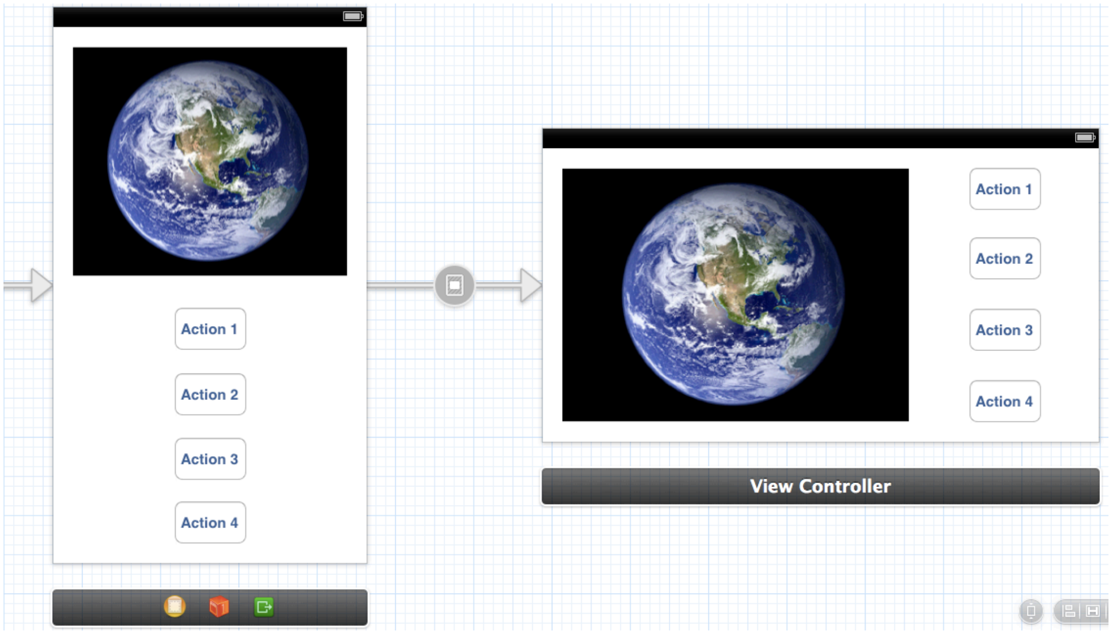
The Segue Indentifier

5. Write the following code in the primary **ViewController.m** file.

```
BOOL isShowingLandscapeView;

@implementation ViewController

//Other setup methods.

- (void)awakeFromNib
{
  isShowingLandscapeView = NO;
  [[UIDevice currentDevice]
beginGeneratingDeviceOrientationNotifications];
  [[NSNotificationCenter defaultCenter] addObserver:self
                             selector:@selector(orientationChanged:)
                                 name:UIDeviceOrientationDidChangeNotification
                               object:nil];
}
```

```
- (void)orientationChanged:(NSNotification *)notification
{
  UIDeviceOrientation deviceOrientation = [UIDevice
currentDevice].orientation;
  if (UIDeviceOrientationIsLandscape(deviceOrientation) && !
isShowingLandscapeView)
  {
    [self performSegueWithIdentifier:@"DisplayAlternateView"
sender:self];
    isShowingLandscapeView = YES;
  }
  else if (UIDeviceOrientationIsPortrait(deviceOrientation) &&
isShowingLandscapeView)
  {
    [self dismissViewControllerAnimated:NO completion:nil];
    isShowingLandscapeView = NO;
  }
}

@end
```

6. Add an `ImageView` and four buttons for each `View`, with a different layout. Running the app in the *Simulator* shows that when an orientation change occurs, the alternate landscape view is presented or dismissed.

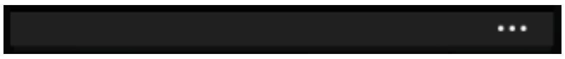


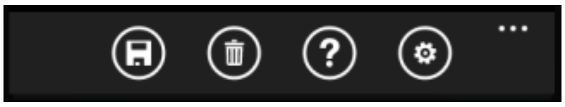
The portrait view and the landscape one

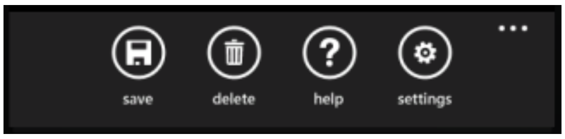
3.4. CHARACTERISTIC ELEMENTS OF WINDOWS PHONE

3.4.1. Application Bar

The *Application Bar* is a Windows Phone feature that gives users quick access to frequently used actions. Using the default Windows Phone App Bar in the development of an app provides consistency in the user experience.
The Application Bar consists in a row of icons and an ellipsis at the bottom of the screen. Clicking the ellipsis shows the buttons labels and a menu item list if available. The App Bar automatically adapts to orientation changes: when the device turns to landscape, the App Bar is shown vertically on the side of the screen, minimizing the waste of space.
There are three different visualizations: the mini size, the default size, and the extended size, which appears after clicking the ellipsis. The following images show each of them, without the appearance of the menu item list.

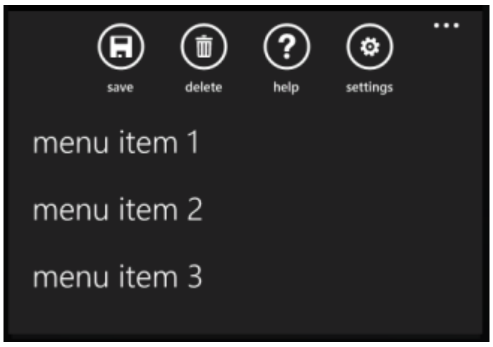

Mini App Bar


Default App Bar



Extended App Bar

If the icon buttons are not enough, the developer can add a small list that is shown when the ellipsis is clicked.



Menu items

This list displays one or more text-based menu items, representing actions less important than the icon buttons ones, or more difficult to explain in a visual way. By default, the text of the menu items is set to lower case, and they are not organized in a hierarchical way: each of the items is independent from the others.

XAML is the suggested way to create an Application Bar, because sample XAML code is already available in the page templates.

```
XAML
<!--Sample code showing usage of ApplicationBar-->
<!--<phone:PhoneApplicationPage.ApplicationBar>
    …
</phone:PhoneApplicationPage.ApplicationBar>-->
```

The developer is required to uncomment the sample code and to add the necessary buttons and items. The following example shows the XAML code for an App Bar with 2 icon buttons and 2 menu items:

```
XAML
<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
    <shell:ApplicationBarIconButton IconUri="/Images/btn1.png"
Text="Button1"/>
    <shell:ApplicationBarIconButton IconUri="/Images/btn2.png"
Text="Button2"/>
    <shell:ApplicationBar.MenuItems>
      <shell:ApplicationBarMenuItem Text="MenuItem 1"/>
      <shell:ApplicationBarMenuItem Text="MenuItem 2"/>
    </shell:ApplicationBar.MenuItems>
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Then the developer can set the Application Bar properties, remove or add more icon buttons, set the icon images and the buttons or menu items texts.
The icon buttons and menu items obviously support click events that must be handled by the code as seen in section 3.1. The following example shows that when the user click the Save icon button, the `Save_Click` method, implemented in C#, is called.

```
XAML
<shell:ApplicationBarIconButton Click="Save_Click" IconUri="/Images/
save.png" Text="Save"/>
```

The Application Bar can be created also by using only C# code. The following are the main steps to take for customizing one.

1. Open the code-behind file for your page in the editor.
2. Initialize the App Bar object adding the following statement after the call to `InitializeComponent`.

```
C#
using Microsoft.Phone.Shell;
public MainPage()
{
    InitializeComponent();
    ApplicationBar = new ApplicationBar();
}
```

3. Set the default mode and other properties.

```
C#
ApplicationBar.Mode = ApplicationBarMode.Default;
ApplicationBar.Opacity = 1.0;
ApplicationBar.IsVisible = true;
ApplicationBar.IsMenuEnabled = true;
```

4. Create the desired number of icon buttons, set the icon images and button text, and then add them to the Application Bar. Do the same for optional menu items.

```
C#
ApplicationBarIconButton button1 = new ApplicationBarIconButton();
button1.IconUri = new Uri("/Images/YourImage.png", UriKind.Relative);
button1.Text = "button 1";
ApplicationBar.Buttons.Add(button1);

ApplicationBarMenuItem menuItem1 = new ApplicationBarMenuItem();
menuItem1.Text = "menu item 1";
ApplicationBar.MenuItems.Add(menuItem1);
```

5. For each icon button and menu item, identify the event to call when the user clicks, creating a new `EventHandler`.

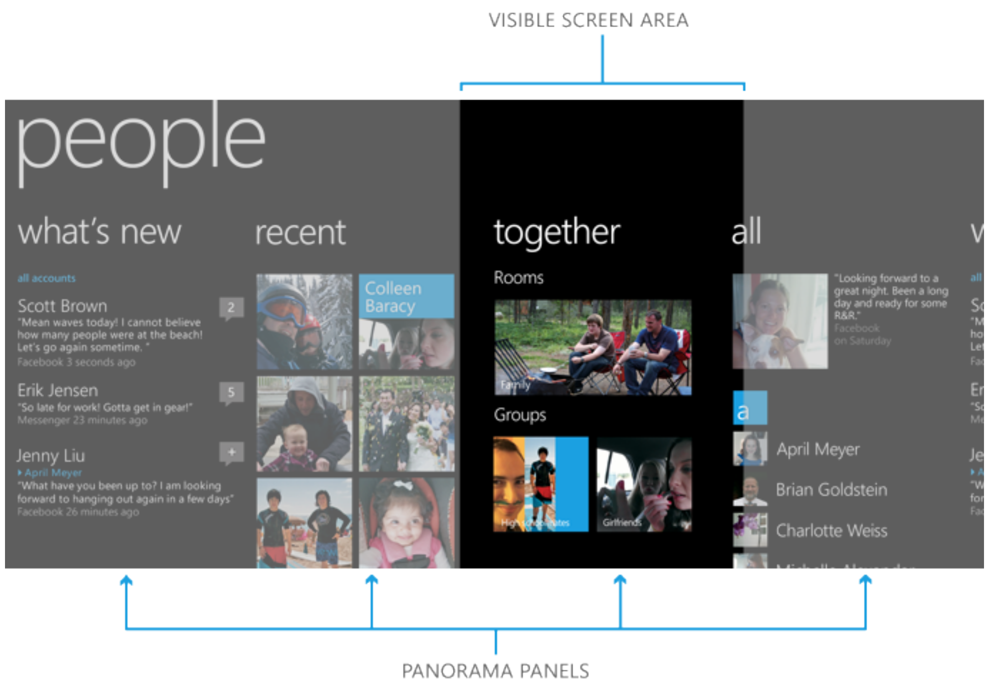
```
C#
button1.Click += new EventHandler(button1_Click);
menuItem1.Click += new EventHandler(menuItem1_Click);
```

6. Finally, specify what to do when the user clicks.

```
C#
private void button1_Click(object sender, EventArgs e)
{  //Do work for your application here.  }

private void menuItem1_Click(object sender, EventArgs e)
{  //Do work for your application here.  }
```

3.4.2. Panorama Control

A panoramic app is characterized by an horizontal expansion, beyond the screen borders, and it is a peculiarity of Windows Phone interface.



Scott Brown 2
"Mean waves today! I cannot believe how many people were at the beach! Let's go again sometime. "
Facebook 3 seconds ago

Erik Jensen 5
"So late for work! Gotta get in gear!"
Messenger 23 minutes ago

Jenny Liu
▸ April Meyer
"What have you been up to? I am looking forward to hanging out again in a few days"
Facebook 26 minutes ago

"Looking forward to a great night. Been a long day and ready for some R&R."
Facebook
on Saturday

Panoramic app

It is managed via a `Panorama` control, that constitutes the base for the app and shows the background image and the title, and multiple `PanoramaItem` controls, that represent the various sections and host the contents. Other elements that can be present or not are the titles of the sections ("items"). Navigation through the sections is possible with horizontal scrolling, and gestures are implemented by default, like for other hosted controls such as lists.

There are two main ways for creating a panorama experience: starting directly from the template named *Windows Phone Panorama App*, or adding a *Windows Phone Panorama Page* to an existing project.

The following example will show how to implement the second solution.

1. After opening an existing project, right-click it in *Solution Explorer*, and add a new item called *Windows Phone Panorama Page*, with the default name **PanoramaPage1.xaml**.
2. This page can be launched from the *MainPage* by an hyperlink, or can be set as the first page when the app starts.
3. By default the following code is already present in the **PanoramaPage1.xaml**, which creates a single `Panorama` control and two `PanoramaItem` controls.

```
XAML
<!--LayoutRoot contains the root grid where all other page content is
placed-->
<Grid x:Name="LayoutRoot">
   <controls:Panorama Title="my application">
      <!--Panorama item one-->
      <controls:PanoramaItem Header="item1">
         <Grid/>
      </controls:PanoramaItem>
      <!--Panorama item two-->
      <controls:PanoramaItem Header="item2">
         <Grid/>
      </controls:PanoramaItem>
   </controls:Panorama>
</Grid>
```

4. Set the background image by adding the following code before the `PanoramaItems`.

```
XAML
<!--Assigns a background image to the Panorama control-->
<controls:Panorama.Background>
   <ImageBrush ImageSource="backgroundImage.jpg"/>
</controls:Panorama.Background>
```

5. It is possible to customize the various parts of the app, changing section orientation and titles, adding controls and so on. The following code adds two `TextBlock` controls in the first `PanoramaItem`.

```
XAML
<!--Panorama item one-->
<controls:PanoramaItem Header="item1">
  <Grid>
    <!--This code places the two TextBlock controls in a StackPanel-->
    <StackPanel>
      <TextBlock Text="This is a text added to the first panorama item"
                 Style="{StaticResource PhoneTextLargeStyle}"
                 TextWrapping="Wrap"/>
      <TextBlock Text=" "/>
      <TextBlock Text="You can put any content you want here..."
                 Style="{StaticResource PhoneTextLargeStyle}"
                 TextWrapping="Wrap"/>
    </StackPanel>
  </Grid>
</controls:PanoramaItem>
```

6. For the second `PanoramaItem`, change the orientation and assign a border and a background.

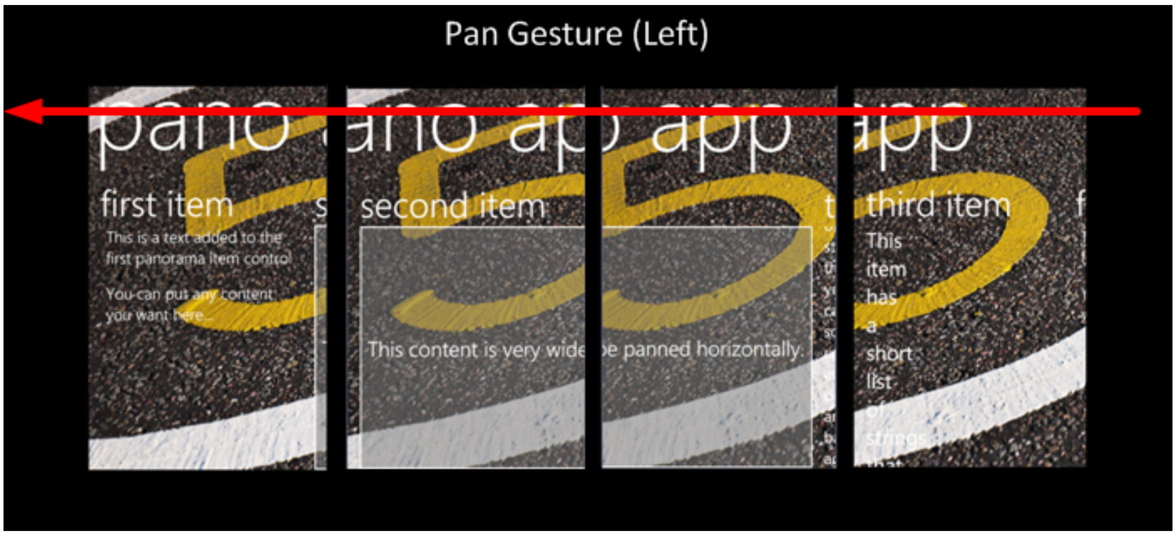
```
XAML
<!--Panorama item two-->
<controls:PanoramaItem Header="item2" Orientation="Horizontal">
  <!--Assigns a border and a background for the content section-->
  <Grid>
```

```
    <Border BorderBrush="{StaticResource PhoneForegroundBrush}"
            BorderThickness="{StaticResource PhoneBorderThickness}"
            Background="#80808080">
      <TextBlock Text="This content is very wide and can be panned
                     horizontally."
                 Style="{StaticResource PhoneTextExtraLargeStyle}"
                 HorizontalAlignment="Center"
                 VerticalAlignment="Center" >
      </TextBlock>
    </Border>
  </Grid>
</controls:PanoramaItem>
```

7. The following code shows how to add a third `PanoramaItem` which contains a `ListBox` control. As seen before, scrolling the list is already supported by default.

```
XAML
<!--This assembly permits to add multiple lines of text to the ListBox->
xmlns:sys="clr-namespace:System;assembly=mscorlib"

<!--Panorama item three-->
<controls:PanoramaItem Header="item3">
  <!--This code adds a series of string text values.-->
  <Grid>
    <ListBox FontSize="{StaticResource PhoneFontSizeLarge}">
      <sys:String>This</sys:String>
      <sys:String>item</sys:String>
      <sys:String>has</sys:String>
      <!--......... -->
      <sys:String>back</sys:String>
      <sys:String>again.</sys:String>
    </ListBox>
  </Grid>
</controls:PanoramaItem>
```

Running the simulator, the resulting app should resemble this illustration:


Pan Gesture (Left)
pano app
first item
This is a text added to the first panorama item control
You can put any content you want here...
second item
This content is very wide and can be panned horizontally.
third item
This
item
has
a
short
list
of
strings

The result of the Panorama example

3.5. CONCLUSIONS

Both iOS and Windows Phone have a powerful IDE that supports developers in their work. Xcode and Visual Studio have lots of similarities, offering the standard set of development tools, but they present also some important differences.
When creating a new project, in Visual Studio it is required to select a programming language, between C#, C++ and Visual Basic, while Objective-C is the only choice for iOS. Differently from other programming languages, Objective-C separates the interface and the implementation of a class in two files, suffixed **.h** and **.m**.
As seen before, both IDEs have a design editor that helps to create the UI, giving an immediate visual result. The important difference is that Visual Studio shows explicitly the layout and binding information in a declarative and easy-to-understand language. Its name is XAML (Extensible Application Markup Language), and it describes everything that is displayed in the visual editor. Therefore, developers have another way for editing the UI, not only selecting and dragging controls with the mouse, but also operating directly in XAML. XAML permits to set properties and even to initialize objects, although its main utility is to make finer modifications to what is already created in the visual editor. If not using XAML, both Visual Studio and Xcode present a Properties pane that supports several modifications on objects. Also associating event code to controls is similar, although in Visual Studio it is a little easier, requiring fewer steps.
Placing objects on a view requires to follow some layout rules. Although it is still possible to use absolute layouts, which use explicit declaration of x-y coordinates, they have become deprecated as dynamic ones have been introduced. They can afford automatic changes, for example when the orientation turns or when they have to adapt to different screen resolutions. In Visual Studio developers can use the star and auto sizing, which uses proportions to distribute the space among controls. Xcode presents an even more complex tool, Auto layout, that creates many relationships between the object and the view, and among the objects themselves. It is an instrument that is as intuitive to use for simple layouts as powerful for complex ones.
As regards screen orientation, iOS and Windows Phone present a common technique that lets users scroll the view to see the hidden content when turning to landscape. When the orientation change requires a completely different layout, Windows Phone suggests using a grid layout and moving controls in different positions. This is a simple technique, but not as powerful as iOS one. In fact in Xcode developers can design two different interfaces, for the portrait orientation and for the landscape one, and each of them is shown alternatively.
Analyzing these few aspects for creating a user interface, Visual Studio and Xcode do not appear so different, with many similarities that make it difficult to choose the best IDE. Visual Studio relies more on the opportunities given by XAML, while Xcode focuses less on code, helped by its clean and polished visual editor.

Bibliography

(1) "iOS: A visual history" by Dieter Bohn, December 13, 2011 - *http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad*

(2) "iOS 6 Review" by Dan Seifert, September 21, 2012 - *http://www.theverge.com/2012/9/21/3363060/ios-6-review*

(3) "Windows Mobile" - *http://en.wikipedia.org/wiki/Windows_Mobile*

(4) "Windows Phone, 7 - 8" - *http://en.wikipedia.org/wiki/Windows_Phone*

(5) "Windows Phone 8 review" by Verge Staff, October 29, 2012 - *http://www.theverge.com/2012/10/29/3570494/windows-phone-8-review*

(6) "Apple Winning the Patent Wars Is Great for Innovation" by Jesus Diaz, August 27, 2012 - *http://gizmodo.com/5938193/apple-winning-the-patent-wars-is-great-for-innovation*

(7) "iOS Human Interface Guidelines" - *http://developer.apple.com/library/ios/#DOCUMENTATION/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html*

(8) "Windows Phone Design Principles" - *http://dev.windowsphone.com/en-us/design/principles*

(9) "Mobile OS comparison: Windows Phone 8 vs iOS 6.0 vs Android 4.1" by Victor H., June 20, 2012 - *http://www.phonearena.com/news/Mobile-OS-comparison-Windows-Phone-8-vs-iOS-6.0-vs-Android-4.1_id31473*

(10) "Microsoft Windows Phone 8 guide: Are these improvements to a great OS enough?" by Matthew Miller, October 29, 2012 - *http://www.zdnet.com/microsoft-windows-phone-8-guide-are-these-improvements-to-a-great-os-enough-7000006486/*

(11) "Windows Phone 8 vs iOS 6 vs Android 4.0 ICS – Home Screens Compared & Contrasted" by Oliver Haslam, June 23, 2012 - *http://www.redmondpie.com/windows-phone-8-vs-ios-6-vs-android-4.0-ics-home-screens-compared-and-contrasted/*

(12) "Why it's OK for the iPhone to have the same UI, and why it isn't for Windows Phone" by Ray S., September 11, 2012 - *http://www.phonearena.com/news/Why-its-OK-for-the-iPhone-to-have-the-same-UI-and-why-it-isnt-for-Windows-Phone_id34307*

(13) "Overview and review of Windows Phone 8" by Daniel Rubino, October 29, 2012 - *http://www.wpcentral.com/overview-and-review-windows-phone-8*

(14) "A Walkthrough the History of the Metro UI" by Jason Lefevers, July 11, 2011 - *http://www.windowsphonemetro.com/2011/07/11/a-walkthrough-the-history-of-the-metro-ui/*

(15) "Introduction To Designing For Windows Phone 7 And Metro" by Daniela Panful, December 20, 2011 - *http://uxdesign.smashingmagazine.com/2011/12/20/introduction-designing-windows-phone-7-metro/*

(16) "Emulating Microsoft’s Metro Design Language" by Connor Turnbull, August 22, 2012 - *http://webdesign.tutsplus.com/articles/typography-articles/emulating-microsofts-metro-design-language/*

(17) "Is Realistic UI Design Realistic?" by Aaron Weyenberg, October 18, 2010 - *http://aaronweyenberg.com/699/is-realistic-ui-design-realistic*

(18) "Skeuomorphism in Interface Design" by Shaun Cronin, July 13, 2012 - *http://webdesign.tutsplus.com/articles/design-theory/skeuomorphism-in-interface-design/*

(19) "The origins of Metro UI" by Michael H., April 24, 2012 - *http://www.phonearena.com/news/The-origins-of-Metro-UI_id29442*

(20) "User Interface for Windows Phone" - *http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff967556(v=vs.105).aspx*

(21) "iOS Developer Library" - *https://developer.apple.com/library/ios/navigation/*

(22) "iPhone iOS 6 Development Essentials" by Neil Smith, 2012 - *http://www.techotopia.com/index.php/IPhone_iOS_6_Development_Essentials*

(23) "Beginning Auto Layout in iOS 6" by Ray Wenderlich, September 19, 2012 - *http://www.raywenderlich.com/20881/beginning-auto-layout-part-1-of-2*