

Unit-V Windows Mobile App Development

Introduction to windows phone 8, Application life cycle, UI designing and Events, Building, Files and storage, Network communication, Push notification, Background agents, Maps and Locations, Data access and storage, Introduction to silverlight and XML, Running and debugging the app, Deploying and publishing.

1. Introduction to windows phone 8

Microsoft was developing Windows Mobile 7 when it realized that the phone wouldn't be an appealing product for consumers who were starting to get used to iPhone or Android devices. So its developers dropped the project and started from scratch to build a totally new platform: Windows Phone 7. The result was really different from the other competitors: a new user interface, based on a flat design style called Microsoft Design style (once known as Metro); and deep integration with social networks and all the Microsoft services, like Office, SkyDrive, and Xbox.

The current version of the platform (which will be covered in this series) is Windows Phone 8; in the middle, Microsoft released an update called Windows Phone 7.5 that added many new consumer features but, most of all, improved the developer experience by adding many new APIs.

Windows Phone 8 is a fresh start for the platform: Microsoft has abandoned the old stack of technologies used in Windows Phone 7 (the Windows Mobile kernel, Silverlight, XNA) to embrace the new features introduced in Windows 8, like the new kernel, the Windows Runtime, and the native code (C++) support.

1.1 Microsoft has released three updates:

- Update 1 (or GDR1), which added some improvements in Internet Explorer, Wi-Fi connectivity, and messaging experience

- Update 2 (or GDR2), which improved support for Google accounts, Xbox Music, and Skype, added FM radio support, and expanded the availability of the Data Sense application to keep track of the data traffic
- Update 3 (or GDR3), which added support for a new resolution (1080p), driving mode, screen lock orientation, and better storage management, and improved the Bluetooth and Wi-Fi stack

1.2 The Hardware

Windows Phone can run on a wide range of devices, with different form factors and hardware capabilities. However, Microsoft has defined a set of hardware guidelines that all manufacturers need to follow to build a Windows Phone device. In addition, vendors can't customize the user interface or the operating system; all the phones, regardless of the producer, offer the same familiar user experience.

This way, Windows Phone can take the best from both worlds: a wide range of devices means more opportunities, because Windows Phone can run well on cheap and small devices in the same way it works well on high-resolution, powerful phones. A more controlled hardware, instead, makes the lives of developers much easier, because they can always count on features like sensors or GPS.

Here are the key features of a Windows Phone 8 device:

- multi-core processor support (dual core and quad core processors)
- at least 512 MB of RAM (usually 1 GB or 2 GB on high-end devices)
- at least 4 GB of storage (that can be expanded with a Micro SD)
- camera
- motion sensors (accelerometer, gyroscope, compass), optional
- proximity sensor, optional
- Wi-Fi and 3G connection
- GPS

- four supported resolutions: **WVGA** (480 × 800), **WXGA** (768 × 1280), **720p**(720 × 1280), and **1080p** (1080 × 1920)
- three hardware buttons: Back, Start, and Search

1.3 The Windows Runtime

The Windows Runtime is the new API layer that Microsoft introduced in Windows 8, and it's the foundation of a new and more modern approach to developing applications. In fact, unlike the .NET framework, it's a native layer, which means better performance. Plus, it supports a wide range of APIs that cover many of the new scenarios that have been introduced in recent years: geolocation, movement sensors, NFC, and much more. In the end, it's well suited for asynchronous and multi-threading scenarios that are one of the key requirements of mobile applications; the user interface needs to be always responsive, no matter which operation the application is performing.

Under the hood of the operating system, Microsoft has introduced the **Windows Phone Runtime**. Compared to the original Windows Runtime, it lacks some features (specifically, all the APIs that don't make much sense on a phone, like printing APIs), but it adds several new ones specific to the platform (like hub integration, contacts and appointments access, etc.).

1.4 Microsoft introduced three features:

- The XAML stack has been ported directly from Windows Phone 7 instead of from Windows 8. This means that the XAML is still managed and not native, but it's completely aligned with the previous one so that, for example, features like behaviors, for which support has been added only in Windows 8.1, are still available). This way, you'll be able to reuse all the XAML written for Windows Phone 7 applications without having to change it or fix it.
- Thanks to a feature called **quirks mode**, applications written for Windows Phone 7 are able to run on Windows Phone 8 devices without having to apply any change in most cases. This mode is able to translate Windows Phone 7 API calls to the related Windows Runtime ones.

- The Windows Phone Runtime includes a layer called **.NET for Windows Phone**, which is the subset of APIs that were available in Windows Phone 7. Thanks to this layer, you'll be able to use the old APIs in a Windows Phone 8 application, even if they've been replaced by new APIs in the Windows Runtime. This way, you'll be able to migrate your old applications to the new platform without having to rewrite all the code.

Like the full Windows Runtime, Windows Phone 8 has added support for **C++** as a programming language, while the **WinJS layer**, which is a library that allows developers to create Windows Store apps using HTML and JavaScript, is missing. If you want to develop Windows Phone applications using web technologies, you'll have to rely on the **WebBrowser** control (which embeds a web view in the application) and make use of features provided by frameworks like PhoneGap.

1.5 The Development Tools

The official platform to develop Windows Phone applications is **Visual Studio 2012**, although support has also been added to the Visual Studio 2013 commercial versions. The major difference is that while Visual Studio 2012 still allows you to open and create Windows Phone 7 projects, Visual Studio 2013 can only be used to develop Windows Phone 8 applications.

There are no differences between the two versions when we talk about Windows Phone development: since the SDK is the same, you'll get the same features in both environments, so we'll use Visual Studio 2012 as a reference for this series.

To start, you'll need to download the Windows Phone 8 SDK from the official developer portal. This download is suitable for both new developers and Microsoft developers who already have a commercial version of Visual Studio 2012. If you don't already have Visual Studio installed, the setup will install the free Express version; otherwise, it will simply install the SDK and the emulator and add them to your existing Visual Studio installation.

The setup will also install **Blend for Windows Phone**, a tool made by Microsoft specifically for designers. It's a XAML visual editor that makes it easier to create a user interface for a Windows Phone application. If you're a developer, you'll probably spend most of the time

manually writing XAML in the Visual Studio editor, but it can be a valid companion when it comes to more complex things like creating animations or managing the visual states of a control.

To install the Windows Phone 8 SDK you'll need a computer with **Windows 8 Pro** or **Windows 8 Enterprise 64-bit**. This is required since the emulator is based on **Hyper-V**, which is the Microsoft virtualization technology that is available only in professional versions of Windows. In addition, there's a hardware requirement: your CPU needs to support the Second Level Address Translation (**SLAT**), which is a CPU feature needed for Hyper-V to properly run. If you have a newer computer, you don't have to worry; basically all architectures from Intel i5 and further support it. Otherwise, you'll still be able to install and use the SDK, but you'll need a real device for testing and debugging.

1.6 The Emulator

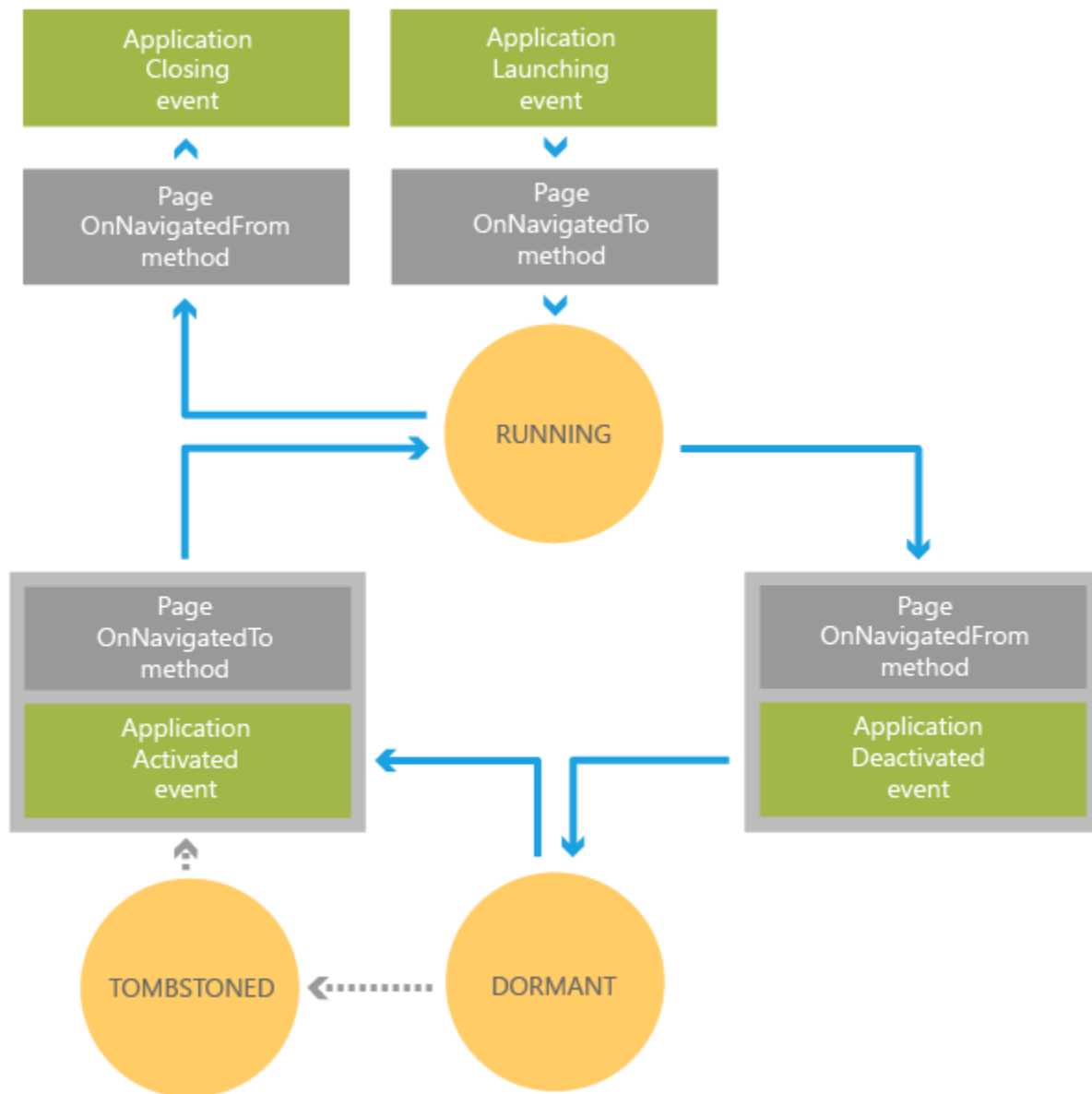
Testing and debugging a Windows Phone app on a device before submitting it to the Windows Phone Store is a requirement; only on a real phone will you be able to test the true performance of the application. During daily development, using the device can slow you down. This is when the emulator is useful, especially because you'll easily be able to test different conditions (like different resolutions, the loss of connectivity, etc.).

The emulator is a virtual machine powered by Hyper-V that is able to interact with the hardware of your computer. If you have a touch monitor, you can simulate the phone touch screen; if you have a microphone, you can simulate the phone microphone, etc. In addition, the emulator comes with a set of additional tools that are helpful for testing some scenarios that would require a physical device, like using the accelerometer or the GPS sensor.

You'll be able to launch the emulator directly from Visual Studio. There are different versions of the emulator to match the different resolutions and memory sizes available on the market.

2. Windows application life cycle

The following image illustrates the lifecycle of a Windows Phone application. In this diagram, the circles are application states. The rectangles show either application- or page-level events where applications should manage their state.



2.1 The Launching Event

A user can launch a new instance of your app by selecting it from the installed applications list or from a Tile on **Start** in addition to other means, such as tapping on a toast

notification associated with the app or selecting the app from the Photos Extras menu. When your app is launched this way, it should present a user interface that makes it clear to the user that a new instance the app was launched. It's ok to provide context about the user's previous experience with the app, such as a list of recent documents the user viewed, but it shouldn't appear as though the user is returning to a previously running instance of the app.

When a new instance of your app is launched, the Launching event is raised. To help ensure that your app loads quickly, you should execute as little code as possible in the handler for this event. In particular, avoid resource-intensive tasks like file and network operations. You should perform these tasks on a background thread after your app has loaded for the best user experience.

2.2 Running

After being launched, an app is Running. It continues to run until the user navigates forward, away from the app, or backwards past the app's first page. Windows Phone apps shouldn't provide a mechanism for the user to quit or exit. Apps also leave the Running state when the phone's lock screen engages unless you have disabled application idle detection. For more information, see Idle detection for Windows Phone 8.

2.3 The OnNavigatedFrom Method

The `OnNavigatedFrom(NavigationEventArgs)` method is called whenever the user navigates away from one of the pages in your app. This can happen as the result of normal page navigation within your application, but it is also called if the user navigates away from your app. Whenever this method is called, your application should store the page state so that it can be restored if the user returns to the page and the page is no longer in memory. The exception to this is backward navigation. The `NavigationMode` property can be used to determine if the navigation is a backward navigation, in which case there is no need to save state because the page will be re-created the next time it is visited.

2.4 The Deactivated Event

The Deactivated event is raised when the user navigates forward, away from your app, by pressing the **Start** button or by launching another application. The **Deactivated** event is also raised if your application launches a Chooser. This event is also raised if the device's lock screen is engaged, unless application idle detection is disabled.

In the handler for the **Deactivated** event, your application should save any unsaved application data so that it can be restored at a later time, if necessary. Windows Phone applications are provided with the State object, which is a dictionary you can use to store application state. If the operating system tombstones your app, as discussed below, it will save this dictionary and return it to you if your app is reactivated..

It is possible for an application to be completely terminated after **Deactivated** is called. When an application is terminated, its state dictionary is not preserved. So you should also store any unsaved state that should be persisted across application instances to isolated storage during the **Deactivated** event.

2.5 Dormant

When the user navigates forward, away from an app, after the **Deactivated** event is raised, the operating system will attempt to put the app into a dormant state. In this state, all of the application's threads are stopped and no processing takes place, but the application remains intact in memory. If the app is reactivated from the dormant, it doesn't need to do anything to re-establish state, because it has been preserved.

If new apps are launched after an app has been made dormant, and these applications requires more memory than is available to provide a good user experience, the operating system will begin to tombstone dormant applications to free up memory.

2.6 Tombstoned

A tombstoned app has been terminated, but the operating system preserves information about its navigation state and also preserves the state dictionaries the app populated during **Deactivated**. The device will maintain tombstoning information for up to five apps at a time. If an app is tombstoned and the user navigates back to the application, it will be relaunched and the application can use the preserved data to restore state.

2.7 The Activated Event

The **Activated** event is called when the user returns to a dormant or tombstoned app. Your app should check the `IsApplicationInstancePreserved` property of the event args to determine whether it is returning from being dormant or tombstoned. If **IsApplicationInstancePreserved** is true, then your app was dormant and state was automatically preserved by the operating system. If it is false, then your app was tombstoned and should use the state dictionary to restore application state

2.8 The OnNavigatedTo Method

The `OnNavigatedTo(NavigationEventArgs)` method is called when the user navigates to a page. This includes when the app is first launched, when the user navigates between the pages of the app, and when the app is relaunched after being made dormant or tombstoned. In this method, your app should check to see whether the page is a new instance. If it is not, then page state does not need to be restored. If the page is a new instance, and there is data in the state dictionary for the page, then you should use this data to restore the state of the page's UI.

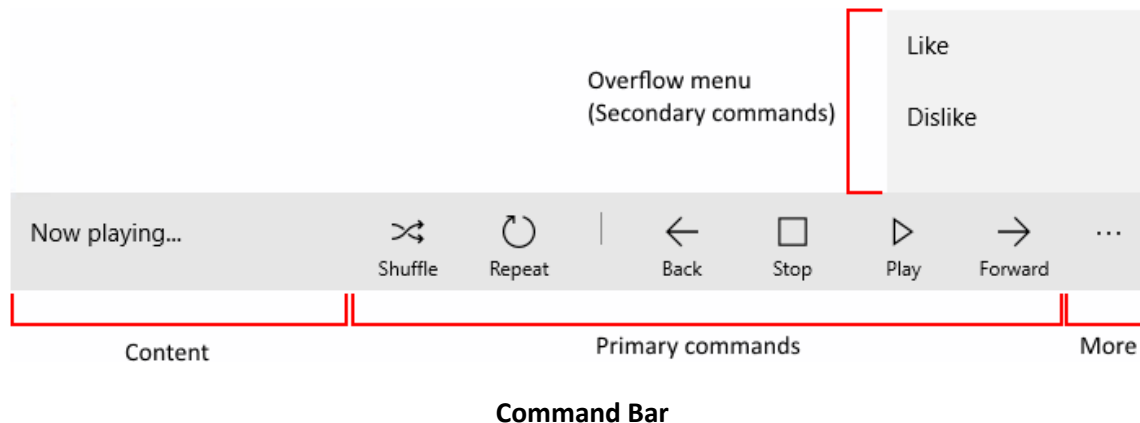
2.9 The Closing Event

The Closing event is raised when the user navigates backwards past the first page of an app. In this case, the app is terminated and no state is saved. In the **Closing** event handler, your app can save data that should persist across instances. There is a limit of 10 seconds for an app to complete all application and page navigation events. If this limit is exceeded, the application is terminated. For this reason, it is a good idea to save persistent state throughout the lifetime of the application and avoid having to do large amounts of file I/O in the **Closing** event handle

3. UI Design Guidelines for Windows Phone 8

3.1 App bar and command bar

Command bars (also called "app bars") provide users with easy access to your app's most common tasks, and can be used to show commands or options that are specific to the user's context, such as a photo selection or drawing mode. They can also be used for navigation among app pages or between app sections. Command bars can be used with any navigation pattern.



The command bar is divided into 4 main areas:

- The "see more" [•••] button is shown on the right of the bar. Pressing the "see more" [•••] button has 2 effects: it reveals the labels on the primary command buttons, and it opens the overflow menu if any secondary commands are present. In the newest SDK, the button will not be visible when no secondary commands and no hidden labels are present. `OverflowButtonVisibility` property allows apps to change this default auto-hide behavior.
- The content area is aligned to the left side of the bar. It is shown if the `Content` property is populated.
- The primary command area is aligned to the right side of the bar, next to the "see more" [•••] button. It is shown if the `PrimaryCommands` property is populated.
- The overflow menu is shown only when the command bar is open and the `SecondaryCommands` property is populated. The new dynamic overflow behavior will automatically move primary commands into the `SecondaryCommands` area when space is limited.

Eg.

```
<CommandBar>
    <AppBarToggleButton Icon="Shuffle" Label="Shuffle" Click="AppBarButton_Click" />
    <AppBarToggleButton Icon="RepeatAll" Label="Repeat" Click="AppBarButton_Click"/>
    <AppBarSeparator/>
    <AppBarButton Icon="Back" Label="Back" Click="AppBarButton_Click"/>
    <AppBarButton Icon="Stop" Label="Stop" Click="AppBarButton_Click"/>
    <AppBarButton Icon="Play" Label="Play" Click="AppBarButton_Click"/>
    <AppBarButton Icon="Forward" Label="Forward" Click="AppBarButton_Click"/>

    <CommandBar.SecondaryCommands>
        <AppBarButton Icon="Like" Label="Like" Click="AppBarButton_Click"/>
        <AppBarButton Icon="Dislike" Label="Dislike" Click="AppBarButton_Click"/>
    </CommandBar.SecondaryCommands>

    <CommandBar.Content>
        <TextBlock Text="Now playing..." Margin="12,14"/>
    </CommandBar.Content>
</CommandBar>
```

Sample Code

3.2 Buttons

A button gives the user a way to trigger an immediate action.

Create the button in XAML.

```
<Button Content="Submit" Click="SubmitButton_Click"/>
```

Or create the button in code.

```
Button submitButton = new Button();

submitButton.Content = "Submit";

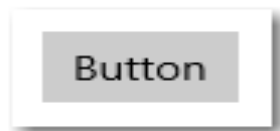
submitButton.Click += SubmitButton_Click;

// Add the button to a parent container in the visual tree.
stackPanel1.Children.Add(submitButton);
```

Handle the Click event.

```
private async void SubmitButton_Click(object sender, RoutedEventArgs e)
{
    // Call app specific code to submit form. For example:
```

```
// form.Submit();  
  
Windows.UI.Popups.MessageDialog messageDialog = new  
Windows.UI.Popups.MessageDialog("Thank you for your submission.");  
  
await messageDialog.ShowAsync();  
  
}
```

**Button**

3.3 Check boxes

A check box is used to select or deselect action items. It can be used for a single item or for a list of multiple items that a user can choose from. The control has three selection states: unselected, selected, and indeterminate. Use the indeterminate state when a collection of sub-choices have both unselected and selected states.

This XAML creates a single check box that is used to agree to terms of service before a form can be submitted.

```
<CheckBox x:Name="termsOfServiceCheckBox" Content="I agree to the terms of  
service."/>
```

the same check box created in code.

```
CheckBox checkBox1 = new CheckBox();  
  
checkBox1.Content = "I agree to the terms of service.";
```

**Check Box**

3.4 Calendar view

A calendar view lets a user view and interact with a calendar that they can navigate by month, year, or decade. A user can select a single date or a range of dates. It doesn't have a picker surface and the calendar is always visible.

The calendar view is made up of 3 separate views: the month view, year view, and decade view. By default, it starts with the month view open.

This example shows how to create a simple calendar view.

```
<CalendarView/>
```

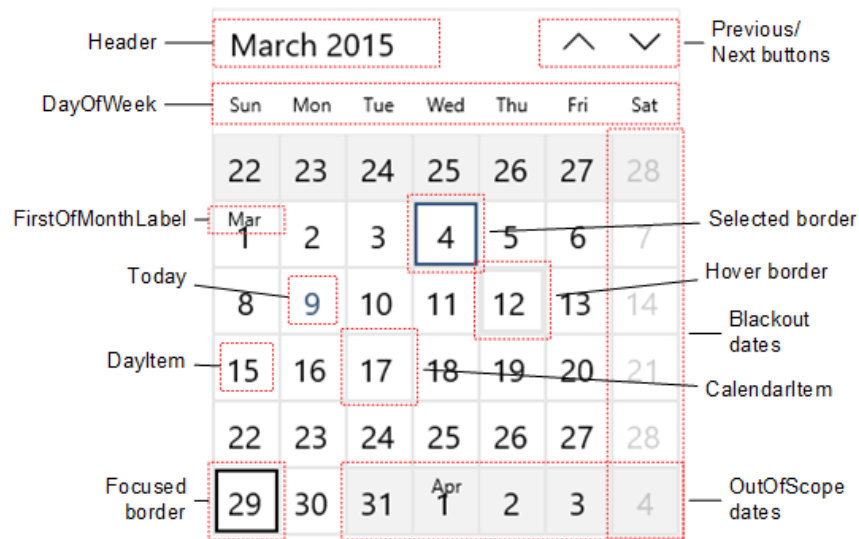
The resulting calendar view looks like this:

September 2014							^	v
Sun	Mon	Tue	Wed	Thu	Fri	Sat		
31	1	2	3	4	5	6		
7	8	9	10	11	12	13		
14	15	16	17	18	19	20		
21	22	23	24	25	26	27		
28	29	30	1	2	3	4		
5	6	7	8	9	10	11		

Calendar View

```
calendarView1.SelectedDates.Add(DateTimeOffset.Now);  
calendarView1.SelectedDates.Add(new DateTime(1977, 1, 5));
```

Customizing the calendar view's appearance



Properties of Calendar View

3.5 Date picker

The date picker gives you a standardized way to let users pick a localized date value using touch, mouse, or keyboard input.

to create a simple date picker with a header.

```
<DatePicker x:Name=birthDatePicker Header="Date of birth"/>
```

or

```
DatePicker birthDatePicker = new DatePicker(); birthDatePicker.Header = "Date of birth";
```

The resulting date picker looks like this:



Date Picker

3.6 Time picker

The time picker gives you a standardized way to let users pick a time value using touch, mouse, or keyboard input.

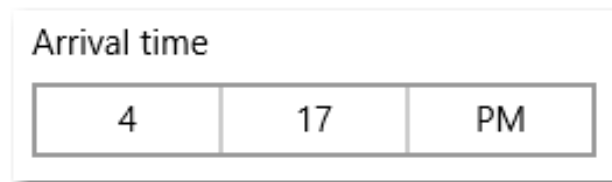
to create a simple time picker with a header.

```
<TimePicker x:Name=arrivalTimePicker Header="Arrival time"/>
```

Or

```
TimePicker arrivalTimePicker = new TimePicker();  
arrivalTimePicker.Header = "Arrival time";
```

The resulting time picker looks like this:



Time Picker

3.7 Dialogs

Dialogs is transient UI elements that appear when something happens that requires notification, approval, or additional information from the user.



Dialog Control

```
private async void displayDeleteFileDialog()  
{
```

```

ContentDialog deleteFileDialog = new ContentDialog()
{
    Title = "Delete file permanently?",
    Content = "If you delete this file, you won't be able to recover it. Do you want to delete it?",
    PrimaryButtonText = "Cancel",
    SecondaryButtonText = "Delete file permanently"
};
ContentDialogResult result = await deleteFileDialog.ShowAsync();
// Delete the file if the user clicked the second button.
// Otherwise, do nothing.
if (result == ContentDialogResult.Secondary)
{
    // Delete the file.
}
}

```

3.8 Hyperlinks

Hyperlinks navigate the user to another part of the app, to another app, or launch a specific uniform resource identifier (URI) using a separate browser app. There are two ways that you can add a hyperlink to a XAML app: the **Hyperlink** text element and **HyperlinkButton** control.

This example shows how to use a Hyperlink text element inside of a [TextBlock](#).

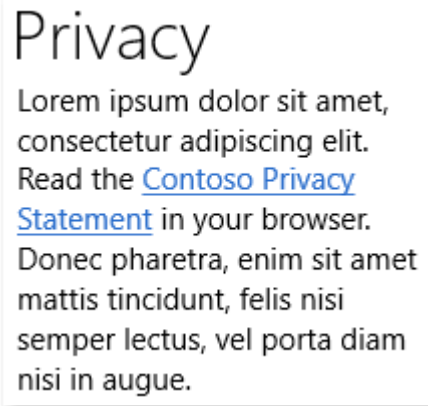
```

<StackPanel Width="200">
    <TextBlock Text="Privacy" Style="{StaticResource SubheaderTextBlockStyle}"/>
    <TextBlock TextWrapping="WrapWholeWords">
        <Span xml:space="preserve"><Run>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Read the
        </Run><Hyperlink NavigateUri="http://www.contoso.com">Contoso Privacy Statement</Hyperlink><Run> in your br
        ower.</Run> Donec pharetra, enim sit amet mattis tincidunt, felis nisi semper lectus, vel porta diam nisi
        in augue.</Span>
    </TextBlock>
</StackPanel>

```

Sample Code

The hyperlink appears inline and flows with the surrounding text:



Hyperlink

3.9 Images

To display an image, you can use either the **Image** object. An Image object renders an image.

to create an image by using the **Image** object.

```
<Image Width="200" Source="licorice.jpg" />
```

Here's the rendered Image object.



image

3.10 Radio buttons

Radio buttons let users select one option from two or more choices. Each option is represented by one radio button; a user can select only one radio button in a radio button group.

```
<StackPanel>
    <StackPanel>
        <TextBlock Text="Background" Style="{ThemeResource BaseTextBlockStyle}" />
        <StackPanel Orientation="Horizontal">
            <RadioButton Content="Green" Tag="Green" Checked="BGRadioButton_Checked" />
            <RadioButton Content="Yellow" Tag="Yellow" Checked="BGRadioButton_Checked" />
            <RadioButton Content="Blue" Tag="Blue" Checked="BGRadioButton_Checked" />
            <RadioButton Content="White" Tag="White" Checked="BGRadioButton_Checked" IsChecked="True" />
        </StackPanel>
    </StackPanel>
    <StackPanel>
        <TextBlock Text="BorderBrush" Style="{ThemeResource BaseTextBlockStyle}" />
        <StackPanel Orientation="Horizontal">
            <StackPanel>
                <RadioButton Content="Green" GroupName="BorderBrush" Tag="Green" Checked="BorderRadioButton_Checked" />
                <RadioButton Content="Yellow" GroupName="BorderBrush" Tag="Yellow" Checked="BorderRadioButton_Checked" IsChecked="True" />
            </StackPanel>
            <StackPanel>
                <RadioButton Content="Blue" GroupName="BorderBrush" Tag="Blue" Checked="BorderRadioButton_Checked" />
                <RadioButton Content="White" GroupName="BorderBrush" Tag="White" Checked="BorderRadioButton_Checked" />
            </StackPanel>
        </StackPanel>
    </StackPanel>
    <Border x:Name="BorderExample1" BorderThickness="10" BorderBrush="#FFFD700" Background="#FFFFFF" Height="50" Margin="0,10,0,10" />
</StackPanel>
```

Sample Code

```
private void BGRadioButton_Checked(object sender, RoutedEventArgs e)
{
    RadioButton rb = sender as RadioButton;

    if (rb != null && BorderExample1 != null)
    {
        string colorName = rb.Tag.ToString();
        switch (colorName)
        {
            case "Yellow":
                BorderExample1.Background = new SolidColorBrush(Colors.Yellow);
                break;
            case "Green":
                BorderExample1.Background = new SolidColorBrush(Colors.Green);
```

```

        break;
    case "Blue":
        BorderExample1.Background = new SolidColorBrush(Colors.Blue);
        break;
    case "White":
        BorderExample1.Background = new SolidColorBrush(Colors.White);
        break;
    }
}
}
private void BorderRadiusButton_Checked(object sender, RoutedEventArgs e)
{
    RadioButton rb = sender as RadioButton;

    if (rb != null && BorderExample1 != null)
    {
        string colorName = rb.Tag.ToString();
        switch (colorName)
        {
            case "Yellow":
                BorderExample1.BorderBrush = new SolidColorBrush(Colors.Gold);
                break;
            case "Green":
                BorderExample1.BorderBrush = new SolidColorBrush(Colors.DarkGreen);
                break;
            case "Blue":
                BorderExample1.BorderBrush = new SolidColorBrush(Colors.DarkBlue);
                break;
            case "White":
                BorderExample1.BorderBrush = new SolidColorBrush(Colors.White);
                break;
        }
    }
}
}

```

The radio button groups look like this.



Radio Buttons

3.11 Text box

The TextBox control lets a user type text into an app. It's typically used to capture a single line of text, but can be configured to capture multiple lines of text. The text displays on the screen in a simple, uniform, plaintext format.

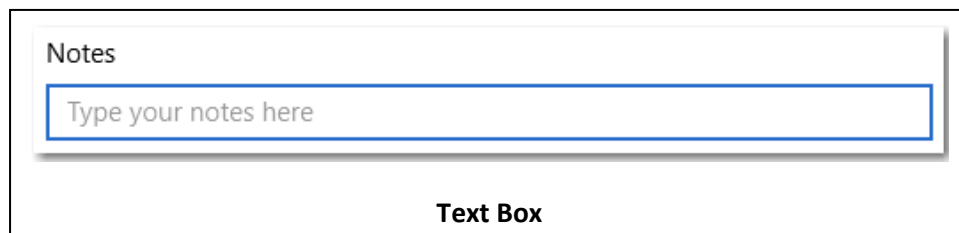
Here's the XAML for a simple text box with a header and placeholder text.

```
<TextBox Width="500" Header="Notes" PlaceholderText="Type your notes here"/>
```

or

```
TextBox textBox = new TextBox();  
textBox.Width = 500;  
textBox.Header = "Notes";  
textBox.PlaceholderText = "Type your notes here";  
  
// Add the TextBox to the visual tree.  
rootGrid.Children.Add(textBox);
```

Here's the text box that results from this XAML.



3.12 Password box

A password box is a text input box that conceals the characters typed into it for the purpose of privacy. A password box looks like a text box, except that it renders placeholder characters in place of the text that has been entered. You can configure the placeholder character.

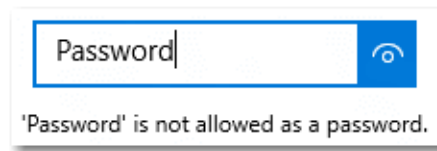
Here's the XAML for a password box control that demonstrates the default look of the PasswordBox.

```
<StackPanel>
```

```
<PasswordBox x:Name="passwordBox" Width="200" MaxLength="16"
    PasswordChanged="passwordBox_PasswordChanged"/>
<TextBlock x:Name="statusText" Margin="10" HorizontalAlignment="Center" />
</StackPanel>

private void passwordBox_PasswordChanged(object sender, RoutedEventArgs e)
{
    if (passwordBox.Password == "Password")
    {
        statusText.Text = "'Password' is not allowed as a password.";
    }
    else
    {
        statusText.Text = string.Empty;
    }
}
```

Here's the result when this code runs and the user enters "Password".



Password

Password character

You can change the character used to mask the password by setting the PasswordChar property. Here, the default bullet is replaced with an asterisk.

```
<PasswordBox x:Name="passwordBox" Width="200" PasswordChar="*" />
```

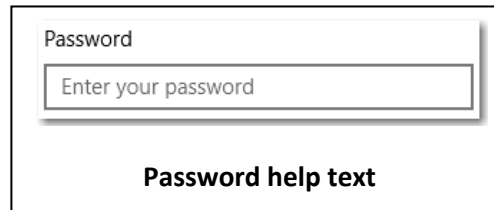
The result looks like this.



Password Character

Headers and placeholder text

```
<PasswordBox x:Name="passwordBox" Width="200" Header="Password"
PlaceholderText="Enter your password"/>
```



3.13 Auto-suggest box

Use an AutoSuggestBox to provide a list of suggestions for a user to select from as they type.

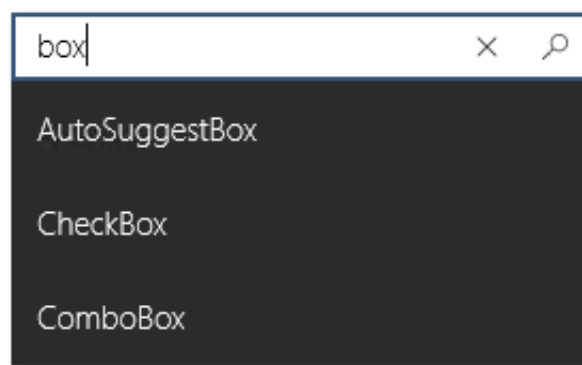
to make the AutoSuggestBox look like a typical search box, add a 'find' icon, like this.

```
<AutoSuggestBox QueryIcon="Find"/>
```

Here's an AutoSuggestBox with a 'find' icon.



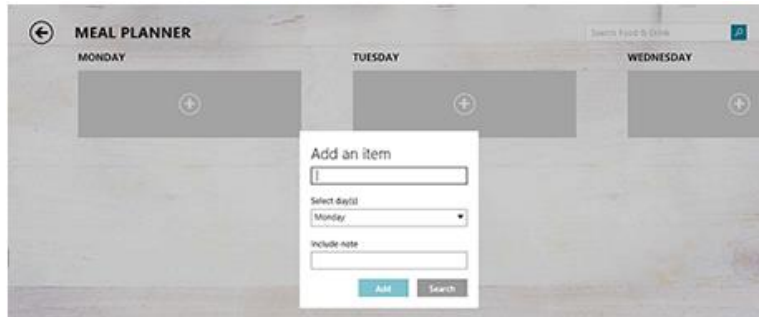
AutoSuggest Box



AutoSuggest Box

3.14 Labels

A label is the name or title of a control or a group of related controls.



Label

3.15 Toggle switches

The toggle switch represents a physical switch that allows users to turn things on or off. Use **ToggleSwitch** controls to present users with exactly two mutually exclusive options (like on/off), where choosing an option results in an immediate action.

This XAML creates the WiFi toggle switch shown previously.

```
<ToggleSwitch x:Name="wiFiToggle" Header="Wifi"/>
```

Here's how to create the same toggle switch in code.

```
ToggleSwitch wiFiToggle = new ToggleSwitch();
```

```
wiFiToggle.Header = "WiFi";
```

```
// Add the toggle switch to a parent container in the visual tree.
```

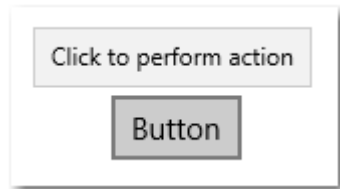
```
stackPanel1.Children.Add(wiFiToggle);
```



Toggle switch

3.16 Tooltips

A tooltip is a short description that is linked to another control or object. Tooltips help users understand unfamiliar objects that aren't described directly in the UI. They display automatically when the user moves focus to, presses and holds, or hovers the mouse pointer over a control. The tooltip disappears after a few seconds, or when the user moves the finger, pointer or keyboard/gamepad focus.



Tooltips

4. Events

An event is a message sent by an object to signal the occurrence of an action. The action could be caused by user interaction, such as touching the screen, or it could be triggered by the internal logic of a class. The object that raises the event is called the event sender. The object that captures the event and responds to it is called the event receiver. Basically the purpose of events is to communicate time-specific, relatively lightweight information from an object at run time, and potentially to deliver that information to other objects in the app.

4.1 Windows Phone events

Generally speaking, Windows Phone events are CLR events, and therefore are events that you can handle with managed code. If you know how to work with basic CLR events already, you have a head start on some of the concepts involved. But you do not necessarily need to know that much about the CLR event model in order to perform some basic tasks, such as attaching handlers.

Because the UI for a typical Windows Phone-based app is defined in markup (XAML), some of the principles of connecting UI events from markup elements to a runtime code entity are similar to other Web technologies, such as ASP.NET, or working with an HTML DOM. In Windows Phone the code that provides the runtime logic for a XAML-defined UI is often referred to as code-behind or the code-behind file. In the Visual Studio solution views, this relationship is shown graphically, with the code-behind file being a dependent and nested file versus the XAML page it refers to.

4.2 Button.Click: an introduction to using Windows Phone events

Generally, you define the UI for your Windows Phone-based app by generating XAML. This XAML can be the output from a designer such as Blend for Visual Studio or from a design surface in a larger IDE such as Windows Phone. The XAML can also be written out in a plaintext editor or a third-party XAML editor. As part of generating that XAML, you can wire event handlers for individual UI elements at the same time that you define all the other attributes that describe that UI element.

If you are using Windows Phone, you can use design features that make it very simple to wire event handlers from XAML and then define them in code-behind. This includes providing an automatic naming scheme for the handlers.

4.3 Defining an event handler

Event handlers in the partial class are written as methods, based on the CLR delegates that are used by that particular event. Your event handler methods can be public, or they can have a private access level. Private access works because the handler and instance created by the XAML are ultimately joined by code generation. The general recommendation is to not make your event handler methods public in the class.

4.4 The sender parameter and event data

Any handler you write for a managed Windows Phone event can access two values that are available as input for each case where your handler is invoked. The first such value is *sender*, which is a reference to the object where the handler is attached. The *sender* parameter is typed as the base **Object** type. A common technique in Windows Phone event handling is to cast *sender* to a more precise type. This technique is useful if you expect to check or change state on the *sender* object itself. Based on your own app design, you expect a type that is safe to cast *sender* to, based on where the handler is attached or other design specifics.

The second value is event data, which generally appears in signatures as the *e* parameter. Per the CLR event model, all events send some kind of event data, with that data captured as an instance of a class that inherits **EventArgs** (or **EventArgs** itself). You can discover which properties for event data are available by looking at the *e* parameter of the delegate that is assigned for the specific event you are handling, and then using Intellisense in Visual Studio or the .NET Framework Class Library for Windows Phone. Some Windows Phone events use the **EventHandler<EventArgs>** delegate or other generic handler types. In most cases, the event definitions constrains the generic with a specific **EventArgs** derived event data class. You should then write the handler method as if it took that **EventArgs** derived event data class directly as the second parameter.

For some events, the event data in the **EventArgs** derived class is as important as knowing that the event was raised. This is especially true of the input events. For keyboard events, key presses on the keyboard raise the same **KeyUp** and **KeyDown** events. In order to

determine which key was pressed, you must access the `KeyEventArgs` that is available to the event handler.

4.5 Adding event handlers in managed code

XAML is not the only way to assign an event handler to an object. To add event handlers to any given object in managed code, including to objects that are not even usable in XAML, you can use the CLR language-specific syntax for adding event handlers. In C#, the syntax is to use the `+=` operator. You instantiate the handler by declaring a new delegate that uses the event handler method name.

If you are using code to add event handlers to objects that appear in the run-time UI, a common practice for Windows Phone is to add such handlers in response to an object lifetime event or callback, such as `Loaded` or `OnApplyTemplate`, so that the event handlers on the relevant object are ready for user-initiated events at run time.

The other option for Visual Basic syntax is to use the **Handles** keyword on event handlers. This technique is appropriate for cases where handlers are expected to exist on objects at load time and persist throughout the object lifetime. Using **Handles** on an object that is defined in XAML requires that you provide a **Name** / **x:Name**. This name becomes the instance qualifier that is needed for the *Instance.Event* part of the **Handles** syntax. In this case you do not need an object lifetime-based event handler to initiate attaching the other event handlers; the **Handles** connections are created when you compile your XAML page.

VB

```
Sub textBlock1_MouseEnter(ByVal sender As Object, ByVal e As MouseEventArgs) Handles  
textBlock1.MouseEnter  
'....  
End Sub  
Sub textBlock1_MouseLeave(ByVal sender As Object, ByVal e As MouseEventArgs) Handles  
textBlock1.MouseLeave  
'....  
End Sub
```

4.6 Routed events

Windows Phone supports the concept of a routed event for several input events that are defined in base classes and are present on most UI elements that support user interaction and input. The following is a list of input events that are routed events:

- `KeyDown`
- `KeyUp`
- `GotFocus`
- `LostFocus`

- `MouseLeftButtonDown`
- `MouseLeftButtonUp`
- `MouseMove`
- `BindingValidationError`

A routed event is an event that is potentially passed on (routed) from a child object to each of its successive parent objects in the object tree. The object tree in question is approximated by the XAML structure of your UI, with the root of that tree being the root element in XAML. The true object tree might vary somewhat from the XAML because the object tree does not include XAML language features such as property element tags.

4.7 The `OriginalSource` property of `RoutedEventArgs`

When an event bubbles up an event route, *sender* is no longer the same object as the event-raising object. Instead, *sender* is the object where the handler that is being invoked is attached. In many cases, *sender* is not the object of interest, and you are instead interested in knowing information such as object held focus when a keyboard key was pressed. In such a case, the value of the `OriginalSource` property is the object of interest.

At all points on the route, `OriginalSource` reports the original object that raised the event, instead of where the handler is attached. For an example scenario where this is useful, consider an app where you want certain key combinations to be "hot keys" or accelerators, regardless of which control currently holds keyboard focus and initiated the event. In terms of the object tree, the focused object might be nested within some items list in a list box, or could be one of hundreds of objects in the overall UI.

4.8 The `Handled` property

Several event data classes for specific routed events contain a property named **Handled**. For examples, see `MouseButtonEventArgs.Handled`, `KeyEventArgs.Handled`, `DragEventArgs.Handled`, and `ValidationErrorEventArgs.Handled`. **Handled** is a settable Boolean property.

Setting the **Handled** property to **true** influences the event system in Windows Phone. When you set the value to **true** in event data, the routing stops for most event handlers; the event does not continue along the route to notify other attached handlers of that particular event case. What "handled" as an action means in the context of the event and how your app responds is up to you. However, you should keep in mind the behavior of the Windows Phone event system if you set **Handled** in your event handlers.

4.8 Input event handlers in controls

Specific existing Windows Phone controls sometimes use this **Handled** concept for input events internally. This can give the appearance from user code that an input event never occurs.

For example, the **Button** class includes logic that deliberately handles the general input event **MouseLeftButtonDown**. Reference topics for specific control classes in the .NET Framework Library often note the event handling behavior implemented by the class. In some cases, the behavior can be changed or appended in subclasses by overriding **OnEvent** methods. For example, you can change how your **TextBox** derived class reacts to key input by overriding **TextBox.OnKeyDown**.

4.9 Registering handlers for already-handled routed events

Earlier it was stated that setting **Handled** to **true** prevented most handlers from acting. The API **AddHandler** provides a technique where you can attach a handler that will always be invoked for the route, even if some other handler earlier in the route has set **Handled** to **true**. This technique is useful if a control you are using has handled the event in its internal compositing or for control-specific logic but you still want to respond to it on a control instance, or higher in the route. However, this technique should be used with caution because it can contradict the purpose of **Handled** and possibly violate a control's intended usage or object model.

4.10 User-initiated events

Windows Phone enforces that certain operations are only permitted in the context of a handler that handles a user-initiated event. The following is a list of such operations:

- Navigating from a **HyperlinkButton**.
- Accessing the primary **Clipboard** API.

Windows Phone user-initiated events include the mouse events (such as **MouseLeftButtonDown**), and the keyboard events (such as **KeyDown**). Events of controls that are based on such events are also considered user-initiated.

API calls that require user initiation should be called as soon as possible in an event handler. This is because the Windows Phone user initiation concept also requires that the calls occur within a certain time window after the event occurrence. In Windows Phone, this time window is approximately one second.

4.11 Removing event handlers

In some circumstances, you might want to remove event handlers during the app lifetime. To remove event handlers, you use the CLR-language-specific syntax. In C#, you use the **=** operator. In Visual Basic, you use the **RemoveHandler** function. In either case, you reference the event handler method name

DATA ACCESS AND STORAGE

Local Storage

The Internet plays an important role in mobile applications. Most Windows Phone applications available in the Store make use of the network connection offered by every device. However, relying only on the network connection can be a mistake; users can find themselves in situations where no connection is available. In addition, data plans are often limited, so the fewer network operations we do, the better the user experience is.

Windows Phone offers a special way to store local data called **isolated storage**. It works like a regular file system, so you can create folders and files as on a computer hard drive. The difference is that the storage is isolated—only your applications can use it. No other applications can access your storage, and users are not able to see it when they connect their phone to the computer. Moreover, as a security measure, the isolated storage is the only storage that the application can use. You're not allowed to access the operating system folders or write data in the application's folder.

Local storage is one of the features which offers duplicated APIs—the old Silverlight ones based on the `IsolatedStorageFile` class and the new Windows Runtime ones based on the `LocalFolder` class. As mentioned in the beginning of the series, we're going to focus on the Windows Runtime APIs.

Working With Folders

The base class that identifies a folder in the local storage is called `StorageFolder`. Even the root of the storage (which can be accessed using the `ApplicationData.Current.LocalStorage` class that is part of the `Windows.Storage` namespace) is a `StorageFolder` object.

This class exposes different asynchronous methods to interact with the current folder, such as:

- `CreateFolderAsync()` to create a new folder in the current path.
- `GetFolderAsync()` to get a reference to a subfolder of the current path.
- `GetFoldersAsync()` to get the list of folders available in the current path.
- `DeleteAsync()` to delete the current folder.
- `RenameAsync()` to rename a folder.

In the following sample, you can see how to create a folder in the local storage's root:

```
private async void OnCreateFolderClicked(object sender, RoutedEventArgs e)
{
    await

    ApplicationData.Current.LocalFolder.CreateFolderAsync("myFolder");
}
```

Working With File

Files, instead, are identified by the `StorageFile` class, which similarly offers methods to interact with files:

- `DeleteAsync()` to delete a file.
- `RenameAsync()` to rename a file.
- `CopyAsync()` to copy a file from one location to another.
- `MoveAsync()` to move a file from one location to another.

The starting point to manipulate a file is the `StorageFolder` class we've previously discussed, since it offers methods to open an existing file (`GetFileAsync()`) or to create a new one in the current folder (`CreateFileAsync()`).

Let's examine the two most common operations: writing content to a file and reading content from a file.

How to Create a File

As already mentioned, the first step to create a file is to use the `CreateFile()` method on a `StorageFolder` object. The following sample shows how to create a new file called `file.txt` in the local storage's root:

```
private async void OnCreateFileClicked(object sender, RoutedEventArgs e)
{
    StorageFile file = await ApplicationData.Current.LocalFolder.CreateFileAsync("file.txt", CreationCollisionOption.ReplaceExisting);
}
```

You can also pass the optional parameter `CreationCollisionOption` to the method to define the behavior to use in case a file with the same name already exists. In the previous sample, the `ReplaceExisting` value is used to overwrite the existing file.

Now that you have a file reference thanks to the `StorageFile` object, you are able to work with it using the `OpenAsync()` method. This method returns the file stream, which you can use to write and read content.

The following sample shows how to write text inside the file:

```
private async void OnCreateFileClicked(object sender, RoutedEventArgs e)
{
    StorageFile file = await ApplicationData.Current.LocalFolder.CreateFileAsync("file.txt", CreationCollisionOption.ReplaceExisting);
    IRandomAccessStream randomAccessStream = await file.OpenAsync(FileAccessMode.ReadWrite);

    using (DataWriter writer = new DataWriter(randomAccessStream.GetOutputStreamAt(0)))
    {
        writer.WriteString("Sample text");
        await writer.StoreAsync();
    }
}
```

The key is the `DataWriter` class, which is a Windows Runtime class that can be used to easily write data to a file. We simply have to create a new `DataWriter` object, passing as a

parameter the output stream of the file we get using the `GetOutputStreamAt()` method on the stream returned by the `OpenAsync()` method.

The `TextWriter` class offers many methods to write different data types, like `WriteDouble()` for decimal numbers, `WriteDateTime()` for dates, and `WriteBytes()` for binary data. In the sample we write text using the `WriteString()` method, and then we call the `StoreAsync()` and `FlushAsync()` methods to finalize the writing operation.

How to Read a File

The operation to read a file is not very different from the writing one. In this case, we also need to get the file stream using the `OpenFile()` method. The difference is that, instead of using the `TextWriter` class, we're going to use the `TextReader` class, which does the opposite operation. Look at the following sample code:

```
private async void OnReadFileClicked(object sender, RoutedEventArgs e)
{
    StorageFile file = await ApplicationData.Current.LocalFolder.GetFilesAsync("file.txt");
    IRandomAccessStream randomAccessStream = await file.OpenAsync(FileAccessMode.Read);

    using (TextReader reader = new TextReader(randomAccessStream.GetInputStreamAt(0)))
    {
        uint bytesLoaded = await reader.LoadAsync((uint) randomAccessStream.Size);
        string readString = reader.ReadString(bytesLoaded);
        MessageBox.Show(readString);
    }
}
```

In this case, instead of the `CreateFileAsync()` method, we use the `GetFileAsync()` method, which can be used to get a reference to an already existing file. Then, we start the reading

procedure using the `DataReader` class, this time using the input stream that we get using the `GetInputStreamAt()` method.

Manage Settings

One common scenario in mobile development is the need to store settings. Many applications offer a Settings page where users can customize different options.

To allow developers to quickly accomplish this task, the SDK includes a class called `IsolatedStorageSettings`, which offers a dictionary called `ApplicationSettings` that you can use to store settings.

Note: The `IsolatedStorageSettings` class is part of the old storage APIs; the Windows Runtime offers a new API to manage settings but, unfortunately, it isn't available in Windows Phone.

Using the `ApplicationSettings` property is very simple: its type is `Dictionary<string, object>` and it can be used to store any object.

In the following sample, you can see two event handlers: the first one saves an object in the settings, while the second one retrieves it.

```
private void OnSaveSettingsClicked(object sender, RoutedEventArgs e)
{
    IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;
    settings.Add("name", "Matteo");
    settings.Save();
}
```

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

```



```
IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;  
if (settings.Contains("name"))  
{  
    MessageBox.Show(settings["name"].ToString());  
}  
}
```

The only thing to highlight is the `Save()` method, which you need to call every time you want to persist the changes you've made. Except for this, it works like a regular `Dictionary` collection.

Debugging the Local Storage

A common requirement for a developer working with local storage is the ability to see which files and folders are actually stored. Since the storage is isolated, developers can't simply connect the phone to a computer and explore it.

The best way to view an application's local storage is by using a third-party tool available on CodePlex called [Windows Phone Power Tools](#), which offers a visual interface for exploring an application's local storage.

The tool is easy to use. After you've installed it, you'll be able to connect to a device or to one of the available emulators. Then, in the **Isolated Storage** section, you'll see a list of all the applications that have been side-loaded from Visual Studio. Each one will be identified by its application ID (which is a GUID). Like a regular file explorer, you can expand the tree structure and analyze the storage's content. You'll be able to save files from the device to your PC, copy files from your PC to the application storage, and even delete items.



Storing Techniques

Serialization and Deserialization

Serialization is the simplest way to store an application's data in the local storage. It's the process that converts complex objects into plain text so that they can be stored in a text file, using XML or JSON as output. Deserialization is the opposite process; the plain text is converted back into objects so that they can be used by the application.

In a Windows Phone application that uses these techniques, serialization is typically applied every time the application's data is changed (when a new item is added, edited, or removed) to minimize the risk of losing data if something happens, like an unexpected crash or a suspension. Deserialization, instead, is usually applied when the application starts for the first time.

Serialization is very simple to use, but its usage should be limited to applications that work with small amounts of data, since everything is kept in memory during the execution. Moreover, it

best suits scenarios where the data to track is simple. If you have to deal with many relationships, databases are probably a better solution (we'll talk more about this later in the article).

In the following samples, we're going to use the same `Person` class we used earlier in this series.

```
public class Person
{
    public string Name { get; set; }
    public string Surname { get; set; }
}
```

We assume that you will have a collection of `Person` objects, which represents your local data:

```
List<Person> people = new List<Person>
{
    new Person
    {
        Name = "Matteo",
        Surname = "Pagani"
    },
    new Person
    {
        Name = "John",
        Surname = "Doe"
    }
};
```

Serialization

To serialize our application's data we're going to use the local storage APIs we learned about in the previous section. We'll use the `CreateFile()` method again, as shown in the following sample:

```
private async void OnSerializeClicked(object sender, RoutedEventArgs e)
{
    DataContractSerializer serializer = new DataContractSerializer(typeof(List<Person>));

    StorageFile file = await ApplicationData.Current.LocalFolder.CreateFileAsync("people.xml");
    IRandomAccessStream randomAccessStream = await file.OpenAsync(FileAccessMode.ReadWrite);

    using (Stream stream = randomAccessStream.AsStreamForWrite())
    {
        serializer.WriteObject(stream, people);
        await stream.FlushAsync();
    }
}
```

The `DataContractSerializer` class (which is part of the `System.Runtime.Serialization` namespace) takes care of managing the serialization process. When we create a new instance, we need to specify which data type we're going to serialize (in the previous sample, it's `List<Person>`). Next, we create a new file in the local storage and get the stream needed to write the data. The serialization operation is made by calling the `WriteObject()` method of the `DataContractSerializer` class, which requires as parameters the stream location in which to write the data and the object to serialize. In this example, it's the collection of `Person` objects we've previously defined.

If you take a look at the storage content using the Windows Phone Power Tools, you'll find a `people.xml` file, which contains an XML representation of your data:

```
<ArrayOfPerson xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/System.Data.DataContractSerializer">
  <Person>
    <Name>Matteo</Name>
    <Surname>Pagani</Surname>
  </Person>
  <Person>
    <Name>John</Name>
    <Surname>Doe</Surname>
  </Person>
</ArrayOfPerson>
```

Deserialization

The deserialization process is very similar and involves, again, the storage APIs to read the file's content and the `DataContractSerializer` class. The following sample shows how to deserialize the data we serialized in the previous section:

```
private async void OnDeserializeClicked(object sender, RoutedEventArgs e)
{
    StorageFile file = await ApplicationData.Current.LocalFolder.GetFilesAsync("people.xml");
    DataContractSerializer serializer = new DataContractSerializer(typeof(List<Person>));

    IRandomAccessStream randomAccessStream = await file.OpenAsync(FileAccessMode.Read);

    using (Stream stream = randomAccessStream.AsStreamForRead())
```

```
{  
    List<Person> people = serializer.ReadObject(stream) as List<Person>;  
}  
}
```

The only differences are:

- We get a stream to read by using the `AsStreamForRead()` method.
- We use the `ReadObject()` method of the `DataContractSerializer` class to deserialize the file's content, which takes the file stream as its input parameter. It's important to note that the method always returns a generic object, so you'll always have to cast it to your real data type (in the sample, we cast it as `List<Person>`).

Using Databases: SQL CE

SQL CE is the database solution that was introduced in Windows Phone 7.5. It's a stand-alone database, which means that data is stored in a single file in the storage without needing a DBMS to manage all the operations.

Windows Phone uses SQL CE 3.5 (the latest release at this time is 4.0, but it is not supported) and doesn't support SQL query execution. Every operation is made using LINQ to SQL, which is one of the first of Microsoft's ORM solutions.

The approach used by SQL CE on Windows Phone is called **code first**. The database is created the first time the data is needed, according to the entities definition that you're going to store in tables. Another solution is to include an already existing SQL CE file in your Visual Studio project. In this case, you'll only be able to work with it in read-only mode.

How to Define the Database

The first step is to create the entities that you'll need to store in your database. Each entity will be mapped to a specific table. entity definition is made using attributes, which are part of the `System.Data.Linq.Mapping` namespace. Each property is decorated with an attribute,

which will be used to translate it into a column. In the following sample we adapt the familiar `Person` class to be stored in a table:

```
[Table]
```

```
public class Person
```

```
{
```

```
    [Column(IsPrimaryKey = true, CanBeNull = false, IsDbGenerated = true)]
```

```
    public string Id { get; set; }
```

```
    [Column]
```

```
    public string Name { get; set; }
```

```
    [Column]
```

```
    public string Surname { get; set; }
```

```
}
```

The entire entity is marked with the `Table` attribute, while every property is marked with the `Column` attribute. Attributes can be customized with some properties, like:

- `IsPrimaryKey` to apply to columns that are part of the primary key.
- `IsDbGenerated` in case the column's value needs to be automatically generated every time a new row is inserted (for example, an automatically incremented number).
- `Name` if you want to assign to the column a different name than the property.
- `DbType` to customize the column's type. By default, the column's type is automatically set by the property's type.

Working With the Database: The DataContext

`DataContext` is a special class that acts as an intermediary between the database and your application. It exposes all the methods needed to perform the most common operations, like insert, update, and delete.

The `DataContext` class contains the connection string's definition (which is the path where the database is stored) and all the tables that are included in the database. In the following sample, you can see a `DataContext` definition that includes the `Person` table we've previously defined:

```
public class DatabaseContext: DataContext
{
    public static string ConnectionString = "Data source=isostore:/Persons.sdf";

    public DatabaseContext(string connectionString):base(connectionString)
    {

    }

    public Table<Person> Persons;
}
```

A separate class of your project inherits from the `DataContext` class. It will force you to implement a public constructor that supports a connection string as its input parameter. There are two connection string types, based on the following prefixes:

- `isostore:/` means that the file is stored in the local storage. In the previous sample, the database's file name is `Persons.sdf` and it's stored in the storage's root.
- `appdata:/` means that the file is stored in the Visual Studio project instead. In this case, you're forced to set the `File Mode` attribute to `Read Only`.

Creating the Database

As soon as the data is needed, you'll need to create the database if it doesn't exist yet. For this purpose, the `DataContext` class exposes two methods:

- `DatabaseExists()` returns whether the database already exists.
- `CreateDatabase()` effectively creates the database in the storage.

In the following sample, you can see a typical database initialization that is executed every time the application starts:

```
private void OnCreateDatabaseClicked(object sender, RoutedEventArgs e)
{
    using (DataContext db = new DataContext(DataContext.ConnectionString))
    {
        if (!db.DatabaseExists())
        {
            db.CreateDatabase();
        }
    }
}
```

Working With the Data

All the operations are made using the `Table<T>` object that we've declared in the `DataContext` definition. It supports standard LINQ operations, so you can query the data using methods like `Where()`, `FirstOrDefault()`, `Select()`, and `OrderBy()`.

In the following sample, you can see how we retrieve all the `Person` objects in the table whose name is Matteo:

```
private void OnShowClicked(object sender, RoutedEventArgs e)
{
    using (DatabaseContext db = new DatabaseContext(DatabaseContext.ConnectionString))
    {
        List<Person> persons = db.Persons.Where(x => x.Name == "Matteo").ToList();
    }
}
```

The returned result can be used not only for display purposes, but also for editing. To update the item in the database, you can change the values of the returned object by calling the `SubmitChanges()` method exposed by the `DataContext` class.

To add new items to the table, the `Table<T>` class offers two methods: `InsertOnSubmit()` and `InsertAllOnSubmit()`. The first method can be used to insert a single object, while the second one adds multiple items in one operation (in fact, it accepts a collection as a parameter).

```
private void OnAddClicked(object sender, RoutedEventArgs e)
{
    using (DatabaseContext db = new DatabaseContext(DatabaseContext.ConnectionString))
    {
        Person person = new Person
        {
            Name = "Matteo",
            Surname = "Pagani"
        };
        db.Persons.InsertOnSubmit(person);
        db.SubmitChanges();
    }
}
```

```
}  
}
```

Please note again the `SubmitChanges()` method: it's important to call it every time you modify the table (by adding a new item or editing or deleting an already existing one), otherwise changes won't be saved.

In a similar way, you can delete items by using the `DeleteOnSubmit()` and `DeleteAllOnSubmit()` methods. In the following sample, we delete all persons with the name Matteo:

```
private void OnDeleteClicked(object sender, RoutedEventArgs e)  
{  
    using (DatabaseContext db = new DatabaseContext(DatabaseContext.ConnectionString))  
    {  
        List<Person> persons = db.Persons.Where(x => x.Name == "Matteo").ToList();  
        db.Persons.DeleteAllOnSubmit(persons);  
        db.SubmitChanges();  
    }  
}
```

Updating the Schema

SQL CE in Windows Phone offers a specific class to satisfy this requirement, called `DatabaseSchemaUpdater`, which offers some methods to update an already existing database's schema.

The key property offered by the `DatabaseSchemaUpdater` class is `DatabaseSchemaVersion`, which is used to track the current schema's version. It's important to properly set it every time we apply an update because we're going to use it when the database is created or updated to recognize whether we're using the latest version.

After you've modified your entities or the `DataContext` definition in your project, you can use the following methods:

- `AddTable<T>()` if you've added a new table (of type `T`).
- `AddColumn<T>()` if you've added a new column to a table (of type `T`).
- `AddAssociation<T>()` if you've added a new relationship to a table (of type `T`).

The following sample code is executed when the application starts and needs to take care of the schema update process:

```
private void OnUpdateDatabaseClicked(object sender, RoutedEventArgs e)
{
    using (DataContext db = new DataContext(DatabaseContext.ConnectionString))
    {
        if (!db.DatabaseExists())
        {
            db.CreateDatabase();

            DatabaseSchemaUpdater updater = db.CreateDatabaseSchemaUpdater();

            updater.DatabaseSchemaVersion = 2;

            updater.Execute();
        }
        else
        {
            DatabaseSchemaUpdater updater = db.CreateDatabaseSchemaUpdater();

            if (updater.DatabaseSchemaVersion < 2)
            {
                updater.AddColumn<Person>("BirthDate");
            }
        }
    }
}
```

```
        updater.DatabaseSchemaVersion = 2;

        updater.Execute();
    }
}
}
```

We're assuming that the current database's schema version is 2. In case the database doesn't exist, we simply create it and, using the `DatabaseSchemaUpdater` class, we update the `DatabaseSchemaVersion` property. This way, the next time the data will be needed, the update operation won't be executed since we're already working with the latest version.

Instead, if the database already exists, we check the version number. If it's an older version, we update the current schema. In the previous sample, we've added a new column to the `Person` table, called `BirthDate` (which is the parameter requested by the `AddColumn<T>()` method). Also in this case we need to remember to properly set the `DatabaseSchemaVersion` property to avoid further executions of the update operation.

In both cases, we need to apply the described changes by calling the `Execute()` method.

SQL Server Compact Toolbox: An Easier Way to Work With SQL CE

Two versions of the tool are available:

- As an [extension](#) that's integrated into commercial versions of Visual Studio.
- As a [stand-alone tool](#) for Visual Studio Express since it does not support extensions.

The following are some of the features supported by the tool:

- Automatically create entities and a `DataContext` class starting from an already existing SQL CE database.

- The generated `DataContext` is able to copy a database from your Visual Studio project to your application's local storage. This way, you can start with a prepopulated database and, at the same time, have write access.
- The generated `DataContext` supports logging in the Visual Studio Output Window so you can see the SQL queries generated by LINQ to SQL.

Advertisement

Using Databases: SQLite

SQLite, from a conceptual point of view, is a similar solution to SQL CE: it's a stand-alone database solution, where data is stored in a single file without a DBMS requirement.

The pros of using SQLite are:

- It offers better performance than SQL CE, especially with large amounts of data.
- It is open source and cross-platform; you'll find a SQLite implementation for Windows 8, Android, iOS, web apps, etc.

SQLite support has been introduced only in Windows Phone 8 due to the new native code support feature (since the SQLite engine is written in native code), and it's available as a Visual Studio extension that you can download from the [Visual Studio website](#).

After you've installed it, you'll find the SQLite for Windows Phone runtime available in the **Add reference** window, in the **Windows Phone Extension** section. Be careful; this runtime is just the SQLite engine, which is written in native code. If you need to use a SQLite database in a C# application, you'll need a third-party library that is able to execute the appropriate native calls for you.

In actuality, there are two available SQLite libraries: **sqlite-net** and **SQLite Wrapper for Windows Phone**. Unfortunately, neither of them is as powerful and flexible as the LINQ to SQL library that is available for SQL CE.

Sqlite-net

Sqlite-net is a third-party library. The original version for Windows Store apps is developed by [Frank A. Krueger](#), while the Windows Phone 8 port is developed by [Peter Huene](#).

The Windows Phone version is available on GitHub. Its configuration procedure is a bit tricky and changes from time to time, so be sure to follow the directions provided by the developer on the [project's home page](#).

Sqlite-net offers a LINQ approach to use the database that is similar to the code-first one offered by LINQ to SQL with SQL CE.

For example, in sqlite-net, tables are mapped with your project's entities. The difference is that, this time, attributes are not required since every property will be automatically translated into a column. Attributes are needed only if you need to customize the conversion process, as in the following sample:

```
public class Person
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; }

    [MaxLength(50)]
    public string Name { get; set; }

    public string Surname { get; set; }
}
```

`Surname` doesn't have any attribute, so it will be automatically converted into a `varchar` column. Instead, we set `Id` as a primary key with an auto increment value, while we specify that `Name` can have a maximum length of 50 characters.

All the basic operations with the database are accomplished using the `SQLiteAsyncConnection` class, which exposes asynchronous methods to create tables, query the data, delete items, etc. It requires as an input parameter the local storage path where the database will be saved.

As with SQL CE and LINQ to SQL, we need to create the database before using it. This is done by calling the `CreateTableAsync<T>()` method for every table we need to create, where `T` is the table's type. In the following sample, we create a table to store the `Person` entity:

```
private async Task CreateDatabase()
{
    SQLiteAsyncConnection conn = new SQLiteAsyncConnection(Path.Combine(ApplicationData.Current.LocalFolder.Path, "Database.db"));
    await conn.CreateTableAsync<Person>();
}
```

In a similar way to LINQ to SQL, queries are performed using the `Table<T>` object. The only difference is that all the LINQ methods are asynchronous.

```
private async void OnReadDataClicked(object sender, RoutedEventArgs e)
{
    SQLiteAsyncConnection conn = new SQLiteAsyncConnection(Path.Combine(ApplicationData.Current.LocalFolder.Path, "Database.db"));
    List<Person> person = await conn.Table<Person>().Where(x => x.Name == "Matteo").ToListAsync();
}
```

In the previous sample, we retrieve all the `Person` objects whose name is Matteo.

Insert, update, and delete operations are instead directly executed using the `SQLiteAsyncConnection` object, which offers the `InsertAsync()`, `UpdateAsync()`, and `DeleteAsync()` methods. It is not required to specify the object's type; sqlite-net will

automatically detect it and execute the operation on the proper table. In the following sample, you can see how a new record is added to a table:

```
private async void OnAddDataClicked(object sender, RoutedEventArgs e)
{
    SQLiteAsyncConnection conn = new SQLiteAsyncConnection(Path.Combine(ApplicationData.Current.LocalCacheFolder.Path, "Database.sqlite"));

    Person person = new Person
    {
        Name = "Matteo",
        Surname = "Pagani"
    };

    await conn.InsertAsync(person);
}
```

Sqlite-net is the SQLite library that offers the easiest approach, but it has many limitations. For example, foreign keys are not supported, so it's not possible to easily manage relationships.

SQLite Wrapper for Windows Phone

SQLite Wrapper for Windows Phone has been developed directly by Microsoft team members (notably Peter Torr and Andy Wigley) and offers a totally different approach than sqlite-net. It doesn't support LINQ, just plain SQL query statements.

The advantage is that you have total control and freedom, since every SQL feature is supported: indexes, relationships, etc. The downside is that writing SQL queries for every operation takes more time, and it's not as easy and intuitive as using LINQ.

The key class is called `Database`, which takes care of initializing the database and offers all the methods needed to perform the queries. As a parameter, you need to set the local storage

path to save the database. If the path doesn't exist, it will be automatically created. Then, you need to open the connection using the `OpenAsync()` method. Now you are ready to perform operations.

There are two ways to execute a query based on the value it returns.

If the query doesn't return a value—for example, a table creation—you can use the `ExecuteStatementAsync()` method as shown in the following sample:

```
private async void OnCreateDatabaseClicked(object sender, RoutedEventArgs e)
{
    Database database = new Database(ApplicationData.Current.LocalFolder, "people.db");

    await database.OpenAsync();

    string query = "CREATE TABLE PEOPLE " +
        "(Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL," +
        "Name varchar(100), " +
        "Surname varchar(100))";

    await database.ExecuteStatementAsync(query);
}
```

The previous method simply executes the query against the opened database. In the sample, we create a `People` table with two fields, `Name` and `Surname`.

The query, instead, can contain some dynamic parameters or return some values. In this case, we need to introduce a new class called `Statement` as demonstrated in the following sample:

```
private async void OnAddDataClicked(object sender, RoutedEventArgs e)
{
    Database database = new Database(ApplicationData.Current.LocalFolder, "people.db");

    await database.OpenAsync();

    string query = "INSERT INTO PEOPLE (Name, Surname) VALUES (@name, @surname)";
    Statement statement = await database.PrepareStatementAsync(query);
    statement.BindTextParameterWithName("@name", "Matteo");
    statement.BindTextParameterWithName("@surname", "Pagani");

    await statement.StepAsync();
}
```

The `Statement` class identifies a query, but it allows additional customization to be performed with it. In the sample, we use it to assign a dynamic value to the `Name` and `Surname` parameters. We set the placeholder using the `@` prefix (`@name` and `@surname`), and then we assign them a value using the `BindTextParameterWithName()` method, passing the parameter's name and the value.

`BindTextParameterWithName()` isn't the only available method, but it's specifically for string parameters. There are other methods based on the parameter's type, such as `BindIntParameterWithName()` for numbers.

To execute the query, we use the `StepAsync()` method. Its purpose isn't just to execute the query, but also to iterate the resulting rows.

In the following sample, we can see how this method can be used to manage the results of a `SELECT` query:

```
private async void OnGetDataClicked(object sender, RoutedEventArgs e)
{
    Database database = new Database(ApplicationData.Current.LocalFolder, "people.db");

    await database.OpenAsync();

    string query = "SELECT * FROM PEOPLE";

    Statement statement = await database.PrepareStatementAsync(query);

    while (await statement.StepAsync())
    {
        MessageBox.Show(statement.GetTextAt(0) + " " + statement.GetTextAt(1));
    }
}
```

NETWORK COMMUNICATION

The Windows Runtime API, [Windows.Networking.Sockets](#), has been adopted for Windows Phone 8. It has been implemented as a Windows Phone Runtime API, making it easy to use in whatever supported programming language you choose. Although we've enhanced the .NET API, [System.Net.Sockets](#), to support more features such as IPv6 and listener sockets, you should consider using the new API for sockets programming because it is more portable than the .NET API. [Windows.Networking.Sockets](#) has been built from the ground up to be clean, secure, and easy-to-use APIs that enforce best practices

New Features in Windows Phone 8

- Two different Networking APIs
- System.Net – Windows Phone 7.1 API, upgraded with new features
- Windows.Networking.Sockets – WinRT API adapted for Windows Phone
- Support for IPV6
- Support for the 128-bit addressing system added to System.Net.Sockets and also is supported in Windows.Networking.Sockets
- NTLM and Kerberos authentication support

- Incoming Sockets
- Listener sockets supported in both System.Net and in Windows.Networking
- Winsock support
- Winsock supported for native development

Windows Phone 8 Emulator and local host

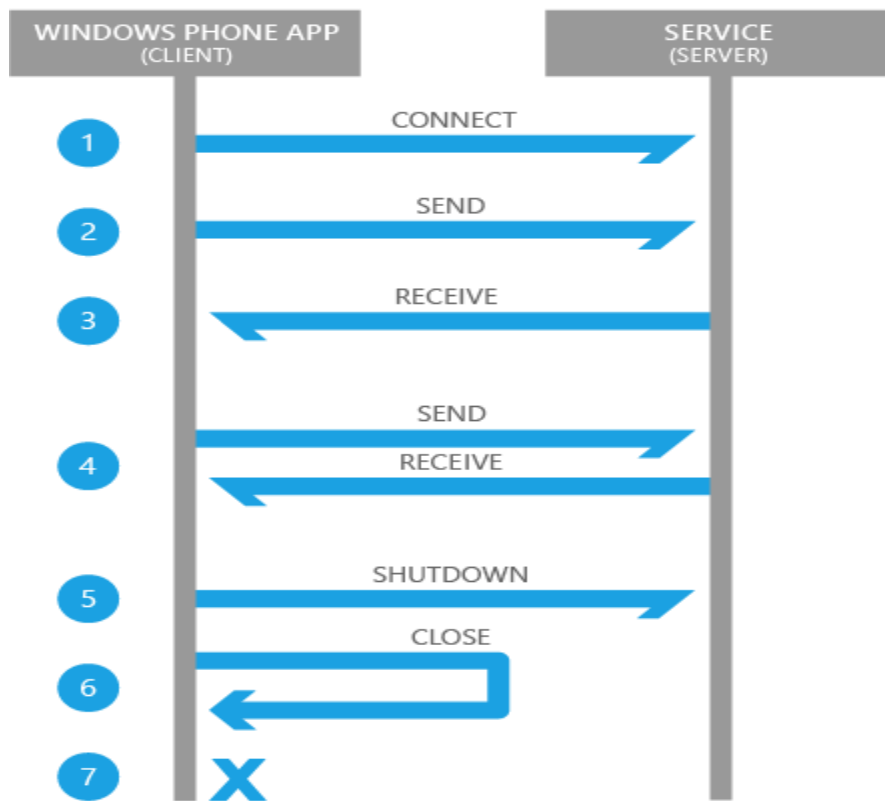
- In Windows Phone 7.x, the emulator shared the networking of the Host PC
- You could host services on your PC and access them from your code using `http://localhost...`
- In Windows Phone 8, the emulator is a Virtual machine running under Hyper-V
- You cannot access services on your PC using `http://localhost...`
- You must use the correct host name or raw IP address of your host PC in URIs

Sockets for Windows Phone 8

Windows Phone provides a programming interface to enable developers to create applications that can communicate with internet services and other remote applications using sockets. Examples of applications and services that use sockets to communicate include FTP, email, chat systems, and streaming multimedia. Using sockets in your Windows Phone application enables you to create rich client applications that can communicate with services over Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) sockets.

Sockets Support on Windows Phone

Windows Phone provides the programming interface needed to create and use TCP and UDP sockets. You can select which type of socket to use based on your application's needs. The following diagram shows a view of the operations that take place during a communication session between a client application and a service.



Operation	TCP	UDP
1	To communicate over TCP, a connection must be established between the client and the server. The endpoint to which the client wants to communicate must be defined as part of the connection request. This is an asynchronous operation in Windows Phone.	Communication over UDP is connectionless, meaning a connection does not have to be created prior to communication.
2	Once the connection has been successfully established, the client can send data to the server by setting up a buffer of data and passing it to the server. TCP is stream-based and the order in which the data is received is guaranteed to be in the order in which it was sent. The TCP protocol takes care of this ordering and reliability for the transmission.	A UDP socket can begin communicating by creating a send request and passing the buffer of data to the server. The successful receipt of the data by the server and the order in which it is received is not guaranteed. If the client requires this certainty, then this must be custom implemented on both the client and the server.
3	The client can request to receive data from the server. This is an asynchronous call and, if successful, the resulting callback will contain the buffer of data that was sent.	A UDP socket can receive data from a service by “listening” on the port associated with this service for any incoming data, and processing it as appropriate.
4	The send and receive pattern in operations 2 and 3 can be repeated for as long as the socket remains connected.	The client can continue to send and receive data.
5	Once the client has finished communicating, it calls shutdown to inform the server that the socket is terminating. This call is used to make sure the remaining data from the server is received before the socket disconnects.	
6	Finally, the client disconnects the socket and closes the communication channel.	
7	At this point, there is no active socket channel, and data sent to the client will be lost.	At this point, there is no active socket channel, and data sent to the client will be lost.

The following is a comparison of the characteristics of TCP and UDP sockets on Windows Phone.

	TCP	UDP
Transmission Type	Stream-based	Datagram
Example Uses	Email, Remote Administration, File Transfer, Web	Streaming Multimedia, Online Games, Internet Telephony
Unicast	Yes	Yes
Any Source Multicast (ASM)	No	Yes
Source-specific Multicast (SSM)	No	Yes
Broadcast	No	No
Connectionless or Connected	Connection-oriented	Connectionless
Reliable Communication	Yes	No

Terminology

A *socket* is a mechanism for delivering data packets or messages between applications or processes. In programming terms, a socket is a programming interface against the TCP/IP protocol stack. Sockets are identified on a network through a socket address, which is a combination of Internet protocol (IP) address and port number. The following table lists some common terminology that you should become familiar with as you work with sockets in your Windows Phone applications.

Term	Description
Broadcast	To send data to all devices on a network.
Client	In socket communication, the consumer of a service provided by a server. For example, a chat client is a consumer of a chat service and can use that service to establish a chat session with other clients. An application running on a Windows Phone device is a client

	application that can consume a service over sockets.
Connectionless	Communication in which a connection does not need to be set up between the sending socket and the receiving socket prior to the communication starting. In this mode, there is no guarantee that the recipient is ready to receive the data and there is also no acknowledgement that the data was ever received, or received with no errors. A UDP socket provides a connectionless communication interface.
Connection-oriented	Communication in which a socket must first set up a connection to a destination socket prior to sending or receiving data. Once a connection is established, a stream of data can be sent and it will be received in the same order. A TCP socket provides a connection-oriented communication interface.
Endpoint	A communication port on either side of the communication. It is typically defined by an IP address, supported transport protocol type, and port number.
IP Address	The industry-standard naming convention for devices on a network. It is a binary number, usually stored in a human readable format such as 172.36.254.14.
IPv4	The older 32-bit addressing system for devices on the Internet. An example of an IPv4 address in human-readable form is 172.36.254.14.
IPv6	<p>The latest 128-bit addressing system for devices on a network. It was developed to accommodate the ever-increasing growth of the number of devices on the Internet. An example of an IPv6 address in human-readable form is fe80::e42b:2e74:6ddb:e30.</p> <p>Important Note:</p> <p>IPv6 is not supported in sockets for Windows Phone OS 7.1.</p>
Multicast	Sending data to devices on a network that have registered interest in that data by joining a multicast group.
Port Number	A number that, combined with an IP address and the transport protocol it supports for communication, identifies a port or endpoint on a network. A well-known list of ports has been reserved for use by specific services such as Telnet (23) and HTTP (80). Other numbers are available for use by other services and applications.
Server	A device on a network that provides a service, or multiple services, for consumption by clients. As an example, a chat server provides a chat service that can be used by chat clients to establish chat sessions with other clients. Although applications on a Windows Phone device can send and receive data over a socket, they are not considered servers.

Socket	A programming interface to communicate with other applications or services on a network.
Transmission Control Protocol (TCP)	An Internet standard that guarantees reliable, in order, delivery of messages on a network.
TCP/IP	The suite of communication protocols used on the Internet and other networks. It was named after the first two protocols that were added to this standard, namely, TCP and IP, but it consists of four layers of protocols, with the TCP and UDP protocols being parts of the Transport Layer.
User Datagram Protocol (UDP)	A Transport Layer protocol used to transmit datagrams in a connectionless manner, meaning that no prior connection needs to be established before sending and receiving messages. This characteristic makes UDP a fast transport protocol, but it can have disadvantages over TCP in terms of reliability since receipt of these datagrams by the destination is not guaranteed and no acknowledgement is sent by default.
Unicast	Sending data to a specific destination with a uniquely identifiable address in a network.

Simple Http Operations – WebClient

```
using System.Net; ... WebClient client; // Constructor
```

```
public MainPage()
```

```
{ ...
```

```
client = new WebClient();
```

```
client.DownloadStringCompleted += client_DownloadStringCompleted;
```

```
}
```

```
void client_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)
```

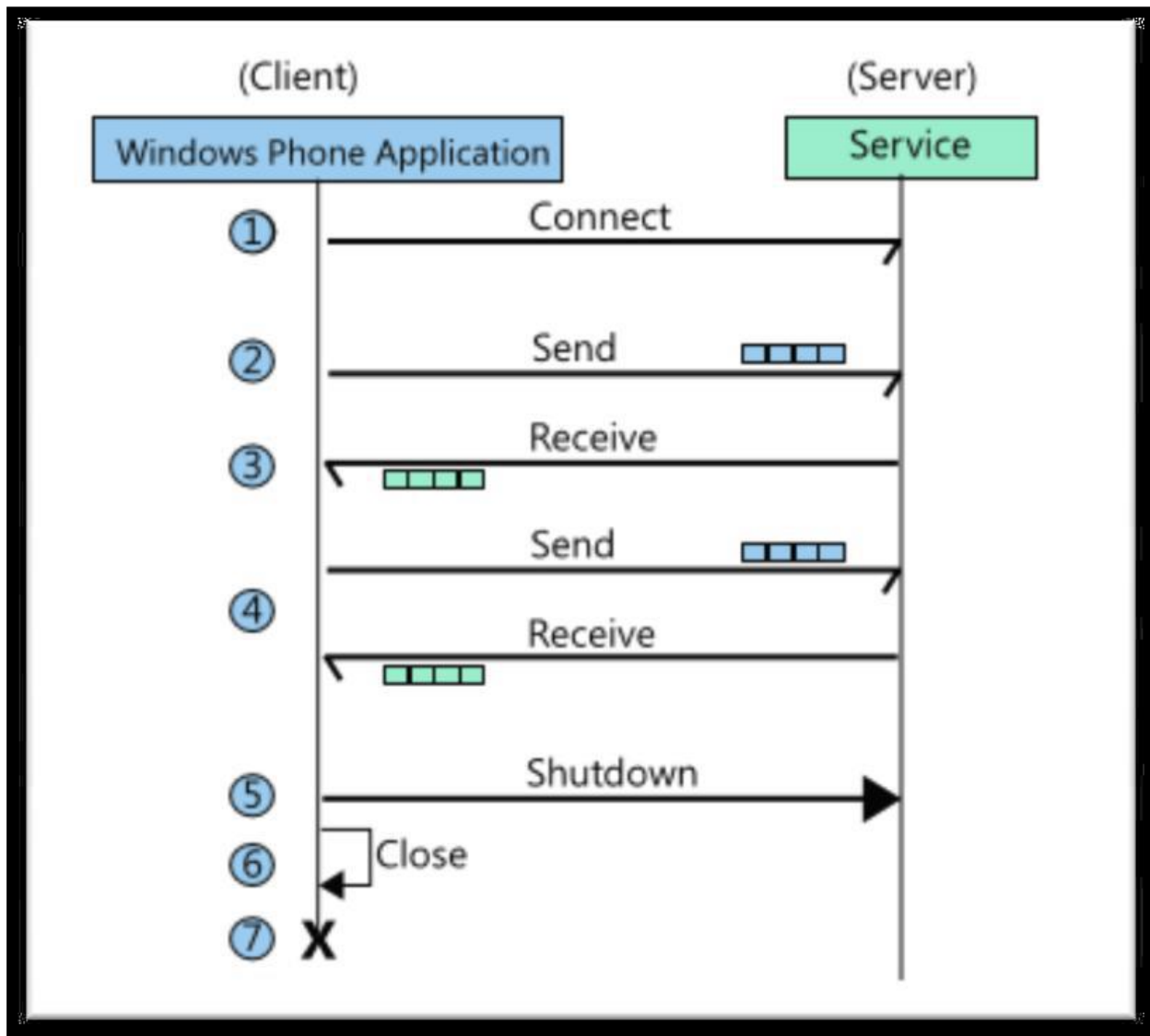
```
{ this.downloadedText = e.Result;
```

```

}

private void loadButton_Click(object sender, RoutedEventArgs e) {
    client.DownloadStringAsync(new Uri("http://MyServer/ServicesApplication/rssdump.xml"));
}

```



Determining the Current Internet Connection Type

```
private const int IANA_INTERFACE_TYPE_OTHER = 1;
```

```
private const int IANA_INTERFACE_TYPE_ETHERNET = 6;
private const int IANA_INTERFACE_TYPE_PPP = 23;
private const int IANA_INTERFACE_TYPE_WIFI = 71; ... string network = string.Empty;
// Get current Internet Connection Profile. ConnectionProfile internetConnectionProfile =
Windows.Networking.Connectivity.NetworkInformation.GetInternetConnectionProfile();

switch (internetConnectionProfile.NetworkAdapter.IanaInterfaceType)

{

case IANA_INTERFACE_TYPE_OTHER: cost += "Network: Other";

break;

case IANA_INTERFACE_TYPE_ETHERNET: cost += "Network: Ethernet";

break;

case IANA_INTERFACE_TYPE_WIFI: cost += "Network: Wifi\r\n";

break;

default:

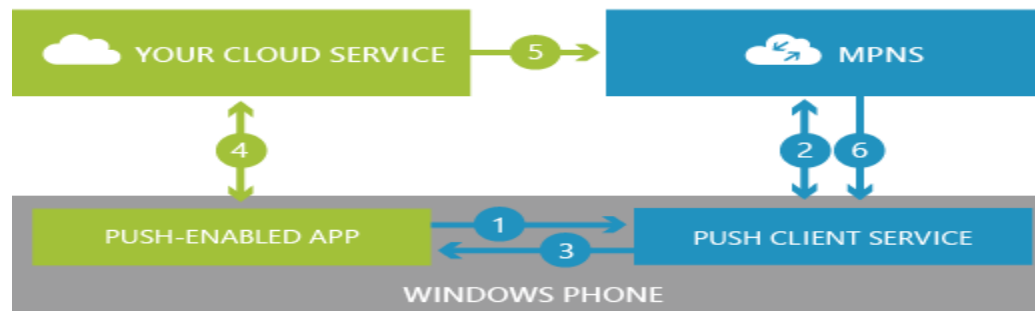
cost += "Network: Unknown\r\n"; break;

}
```

PUSH NOTIFICATION

Microsoft Push Notification Service in Windows Phone is an asynchronous, best-effort service that offers third-party developers a channel to send data to a Windows Phone app from a cloud service in a power-efficient manner.

The following diagram shows how a push notification is sent.



1. Your app requests a push notification URI from the Push client service.
2. The Push client service negotiates with the Microsoft Push Notification Service (MPNS), and MPNS returns a notification URI to the Push client service.
3. The Push client service returns the notification URI to your app.
4. Your app can then send the notification URI to your cloud service.
5. When your cloud service has info to send to your app, it uses the notification URI to send a push notification to MPNS.
6. MPNS routes the push notification to your app.

To send push notifications, your web service or app must:

- Create a POST message for each Windows Phone device to which you want to send a notification.
- Form the message for the appropriate notification type. The following sections describe the message formats for toast, Tile, and raw notification messages. You can post only one notification type (toast, Tile, or raw) to the server at a time. If you want to send multiple notification types to the same client device at the same time, you must create separate POST messages for each notification type.
- Post the messages to the push notification service.
- Get the response from the push notification service and respond accordingly.

Custom HTTP headers

Custom HTTP headers can include a notification message ID, batching interval, the type of push notification being sent, and the notification channel URI. The **MessageID** is the notification message ID associated with the response. If this header is not added to the POST request, the push notification service omits this header in the response.

The header specification is "X-MessageID": "1*MessageIDValue MessageIDValue = **STRING** (uuid).

For example: X-MessageID: *UUID*

The **NotificationClass** is the batching interval that indicates when the push notification will be sent to the app from the push notification service. See the tables in the toast, Tile, and raw notification sections for possible values for this header. If this header is not present, the message will be delivered by the push notification service immediately.

The header specification is "X-NotificationClass": "1*NotificationClassValue NotificationClassValue = **DIGIT**.

For example: X-NotificationClass: 1

The **Notification Type** is the type of push notification being sent. Possible options are Tile, toast, and raw. If this header is not present, the push notification will be treated as a raw notification. For more info, see [Push notifications for Windows Phone 8](#). The header specification is "X-WindowsPhone-Target": "1*NotificationTypeValue NotificationTypeValue = **STRING**.

For example: X-WindowsPhone-Target: toast.

Special characters

The following characters should be encoded as shown in the table when used in a Tile or toast payload.

Character	XML encoding
<	<
>	>
&	&
'	'
"	"

Tile and toast notification payloads

The following sections describe payload info needed for sending a push notification to a toast or Tile.

Toast notification payloads

For general info about how to structure the toast notification payload using code or XML, as well as for info about how to structure the payload to deep link into your app, see [Toasts for Windows Phone 8](#).

Toast notification payload HTTP headers

Use the following HTTP headers when creating a toast notification:

C#

```
sendNotificationRequest.ContentType = "text/xml";  
sendNotificationRequest.Headers.Add("X-WindowsPhone-Target", "toast");  
sendNotificationRequest.Headers.Add("X-NotificationClass", "[batching interval]");
```

Toast notification batching intervals

The following table describes the values that the batching interval can have.

Value	Delivery interval
2	Immediate delivery.
12	Delivered within 450 seconds.
22	Delivered within 900 seconds.

Tile notification batching intervals

The following table describes the values that the batching interval can have.

Value	Delivery interval
1	Immediate delivery.
11	Delivered within 450 seconds.
21	Delivered within 900 seconds.

Sending push notifications to secondary Tiles

If your app has secondary Tiles, the **Id** attribute designates which Tile to update. You can omit the **Id** attribute of the **Tile** element if updating your app's default Tile.

The following code shows an example of the **Id** attribute of the **Tile** element, which should contain the exact navigation URI of the secondary Tile.

```
string tileMessage = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
"<wp:Notification xmlns:wp=\"WPNotification\">" +
  "<wp:Tile Id=\"/SecondaryTile.xaml?DefaultTitle=FromTile\">" +
  ...
  "</wp:Tile>" +
"</wp:Notification>";
```

Raw Tile notification payload

Use the following HTTP headers when sending a raw Tile notification.

C#

```
sendNotificationRequest.ContentType = "text/xml";
sendNotificationRequest.Headers.Add("X-NotificationClass", "[batching interval]");
```

The following table describes the values that the batching interval can have.

Value	Delivery interval
3	Immediate delivery.
13	Delivered within 450 seconds.
23	Delivered within 900 seconds.

The structure of the payload is defined by the app. The following code shows an example.

```
string tileMessage = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
  "<root>" +
    "<Value1>[UserValue1]<Value1>" +
    "<Value2>[UserValue2]<Value2>" +
  "</root>"
```

You can also pass a byte stream. The following code shows an example.

```
new byte[] {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

Background agents for Windows Phone 8

Scheduled Tasks and background agents allow an application to execute code in the background, even when the application is not running in the foreground. The different types of Scheduled Tasks are designed for different types of background processing scenarios and

therefore have different behaviors and constraints. This topic describes the scheduling, duration, and limitations of scheduled tasks.

The following are the types of Scheduled Tasks. Note that [ScheduledTask](#) derives from [ScheduledAction](#). The code that runs in the background is placed in a class that derives from [ScheduledTaskAgent](#), which derives from [BackgroundAgent](#).

Scheduled Task Type	Description
PeriodicTask	<i>Periodic agents</i> run for a small amount of time on a regular recurring interval. Typical scenarios for this type of task include uploading the device's location and performing small amounts of data synchronization.
ResourceIntensiveTask	<i>Resource-intensive agents</i> run for a relatively long period of time when the phone meets a set of requirements relating to processor activity, power source, and network connection. A typical scenario for this type of task is synchronizing large amounts of data to the phone while it is not being actively used by the user.

Background Agent Lifecycle

An application may have only one background agent. This agent can be registered as a **PeriodicTask**, a **ResourceIntensiveTask**, or both. The schedule on which the agent runs depends on which type of task it is registered as. The details of the schedules are described later in this topic. Only one instance of the agent runs at a time.

The code for the agent is implemented by the application in a class that inherits from [BackgroundAgent](#). When the agent is launched, the operating system calls [OnInvoke\(ScheduledTask\)](#). In this method, the application can determine which type of **ScheduledTask** it is being run as, and perform the appropriate actions.

When the agent has completed its task, it should call [NotifyComplete\(\)](#) or [Abort\(\)](#) to let the operating system know that it has completed. **NotifyComplete** should be used if the task was successful. If the agent is unable to perform its task – such as a needed server being unavailable – the agent should call **Abort**, which causes the [IsScheduled](#) property to be set to false.

The following constraints apply to all Scheduled Tasks.

Constraint	Description
------------	-------------

Unsupported APIs	There is a set of APIs that cannot be used by any Scheduled Task. Using these APIs either will cause an exception to be thrown at run time or will cause the application to fail certification during submission to Store. For the list of restricted APIs, see Unsupported APIs for background agents for Windows Phone 8 .
Memory usage cap	Periodic agents and resource-intensive agents can use no more than 20 MB of memory at any time on devices with 1 GB of memory or more.
Reschedule required every two weeks	Use the ExpirationTime property of the ScheduledTask object to set the time after which the task no longer runs. This value must be set to a time within two weeks of the time when the action is scheduled with the Add(ScheduledAction) method.
Agents unscheduled after two consecutive crashes	Both periodic and resource-intensive agents are unscheduled if they exit two consecutive times due to exceeding the memory quota or any other unhandled exception. The agents must be rescheduled by the foreground application.

The following are the schedule, duration, and general constraints for Resource-intensive agents.

Constraint	Description
Duration: 10 minutes	Resource-intensive agents typically run for 10 minutes. There are other constraints that may cause an agent to be terminated early.
External power required	Resource-intensive agents do not run unless the device is connected to an external power source.
Non-cellular connection required	Resource-intensive agents do not run unless the device has a network connection over Wi-Fi or through a connection to a PC.
Minimum battery power	Resource-intensive agents do not run unless the device's battery power is greater than 90%.
Device screen lock required	Resource-intensive agents do not run unless the device screen is locked.
No active phone call	Resource-intensive agents do not run while a phone call is active.

Cannot change network to cellular	If a resource-intensive agent attempts to call AssociateToNetworkInterface(Socket, NetworkInterfaceInfo) specifying either MobileBroadbandGSM() or MobileBroadbandCDMA(), the method call fails.
-----------------------------------	--

Introduction to Silverlight

Microsoft **Silverlight** is a deprecated application framework for writing and running rich Internet applications, similar to Adobe Flash. A plug-in for **Silverlight** is available for some browsers.

Microsoft Silverlight is a cross-browser, cross-platform implementation of .NET for building and delivering the next generation of media experiences & rich interactive applications for the Web.

What is Silverlight?

- Like Flash:
 - Browser plug-in: cross-browser, cross-platform
 - Animated ads, video, applications (like Flex)
 - Benefits:
 - Write-once-run-everywhere,
 - Adds functionality not in HTML / AJAX

Versions of Silverlight

- v 1.0
 - RTM in Sept. 2007
 - Code behind – JavaScript only
- v 1.1 / 2
 - .NET based
 - SL 1.1 Alpha introduced in Spring 2007.
 - SL 2 Beta 1 released at MIX08 in March 2008
 - RTM maybe Q3-2008? For Olympics?
 - Code behind - .NET languages C#, etc.
 - Partial .NET class library

SL for Mobile: Schedule



SL for Mobile: Weatherbug Demo



Inside Silverlight 2

Silverlight 2:

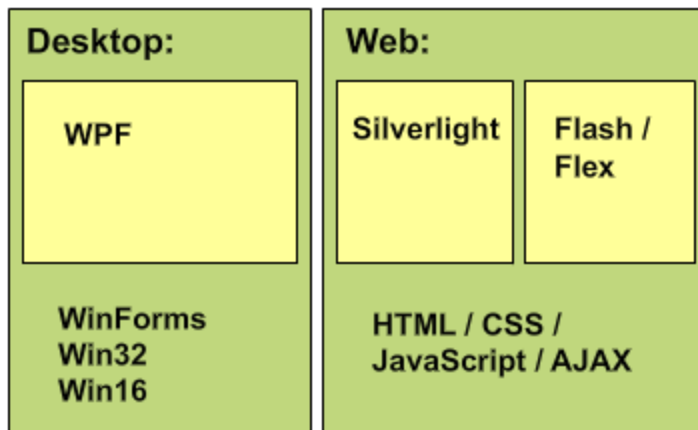
GUI “eye-candy”:

XAML, Layout, Styling, Animation

.NET “plumbing”:

CLR, Base Class Libraries

Comparing client platforms



Competing technologies

- Web-based:
 - Adobe Flash / Flex
 - “Ajax”: HTML + CSS + JavaScript
- Desktop based “smart clients”
 - WPF on high end
 - 3D, Hardware acceleration
 - WinForms: (Mature, proven)
- Desktop-web hybrid (?):
 - Adobe AIR

Silverlight and WPF: Differences

- WPF:
 - Windows only
 - Requires 50 / 200 MB .NET 3.x runtime
 - Steeper learning curve
- Silverlight:
 - Cross OS, cross browser
 - Small download (approx. 4 to 5 MB)
 - Reduced feature set
 - Sandboxed – Secure but limiting

How important is Silverlight?

- This is a big deal
- Once-every-20-years event
- Existing client-side web technology has reached the peak of its life-cycle
- Fresh start of a new client GUI technology
- Web is where the action is
 - The network is the computer

What Can Silverlight Do?

Feature Details

SL2 Feature Summary:

- GUI system features:
 - XAML, etc.
- Controls:
 - What controls come “in the box”?
- Data:
 - Features related to database-type apps
- Communications:
 - Web services

GUI System Features

- WPF subset
- Vector based vs. pixel based
 - Scalable – Looks good at multiple resolutions
- Dynamic layouts
- XAML – Similar to HTML
 - Declarative
 - Designers and programmers work in parallel
- Rich customization is easier
 - “Lookless” controls
 - Styles and templating

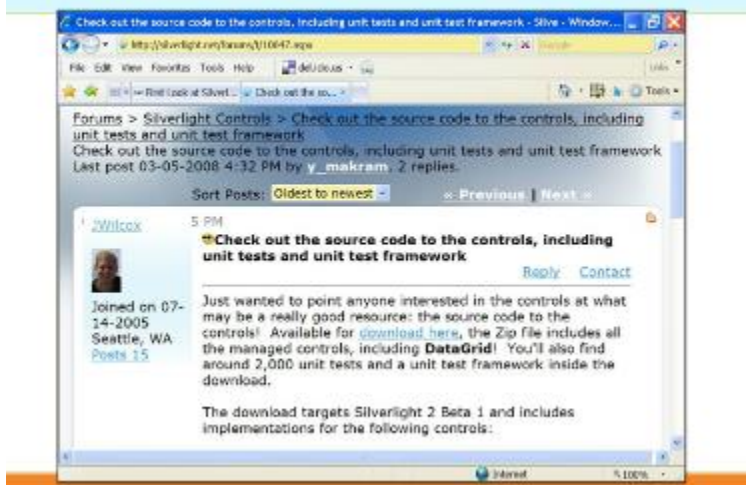
Non-GUI Features

- More than just “eye-candy”
 - OpenFileDialog
 - Threads
 - Direct cross-domain access rather than proxied by your server.

SL2 Features: Controls

- Controls:
 - Extensible control base classes
 - Common controls:
 - Textbox, Checkbox, Radiobutton, etc
 - TabControl, Slider, ScrollViewer, ProgressBar, etc
 - Layout controls:
 - Grid, StackPanel
 - Data controls:
 - DataGrid, etc

SL2 Features: Control Source Code



SL2 Features: Data

- Data:
 - 2-way data binding
 - More LINQ support:
 - LINQ to XML
 - LINQ to Objects

SL2 Features: Communications

- Communications:
 - REST, POX, RSS, and WS-* communication
 - Cross domain network access (coming)
 - Sockets (no cross-domain yet)
 - WCF? (How much client side?)

SL2 Features: Other

- Isolated Storage:
 - Secure
 - Size? (100KB, expandable to X?)
- Security
 - See Perry Birch's talk from 1:30-3:00

Silverlight Development Walk-through

Silverlight Tools

- Expression Blend:
 - For graphic designers
 - GUI builder
- Visual Studio 2008:
 - For programmers
 - Includes a more limited GUI builder

Background Agents

With the release of Windows Phone 7 Mango, you now have the ability to multitask (scheduled multitask) by using background agents. Background agents allow you to do things when your application is not running.

It is important to understand that the OS is responsible for determining when your background agent can run and is determined by a number of factors. It is also dependent on the type of Background Agent you use.

For both types of Agents you are constrained by the following:

- Unsupported APIs
- Memory Cap Usage
- Agent Crashes (unscheduled Agents after two crashes)
- Rescheduling (You have to reschedule every two weeks)

Periodic Agents (PeriodicTask) are used when you want a “semi” predictable action to fire. But they are constrained to the following. For example, you can use Periodic Agents for collecting quick GPS coordinates or updating an RSS feed.

- 30 Minute Intervals (this time may drift)
- Run for 25 seconds
- Might not run on Battery Saver mode
- Agents per device (The number of apps using agents) can be as low as 6 on some devices

Resource-intensive Agents (resourceIntensiveTask) can be used for more intensive items like downloading larger files or coping database entries to a replication server. But you must keep in mind that they have some specific constraints as well.

- Duration 10 Minutes
- External Power Required (You need to plug it in)
- Connection through WiFi or PC
- Battery 90% or better
- Screen Lock on
- No active phone call
- Add a ScheduledAgentTasks project to your solution
- Add a reference to the agent project in your phone application project
- Add your code to the Invoke Method in the ScheduledAgentTasks project as shown below

```
protected override void OnInvoke(ScheduledTask task)
```

```
{
```

```
//TODO: Add code to perform your task in background

string toastMessage = "";

// If your application uses both PeriodicTask and ResourceIntensiveTask
// you can branch your application code here. Otherwise, you don't need to.
if (task is PeriodicTask)
{
    // Execute periodic task actions here.

    toastMessage = "Periodic task running.";
}
else
{
    // Execute resource-intensive task actions here.

    toastMessage = "Resource-intensive task running.";
}


// Launch a toast to show that the agent is running.

// The toast will not be shown if the foreground application is running.
ShellToast toast = new ShellToast();
toast.Title = "Background Agent Sample";
toast.Content = toastMessage;
toast.Show();


// If debugging is enabled, launch the agent again in one minute.
#if DEBUG_AGENT

ScheduledActionService.LaunchForTest(task.Name, TimeSpan.FromSeconds(60));
```

```
#endif
```

```
// Call NotifyComplete to let the system know the agent is done working.
```

```
NotifyComplete();
```

```
}
```

The one caveat is that it can sometimes be difficult to debug in an emulator. If you have a developer phone you will have a much easier time debugging it on the device.

Background Agents

Periodic agents run for a *short time* on a regular *recurring interval*.

Examples:

- Uploading the device's location
- Small amounts of data synchronization

Periodic Agent Constraints

Scheduled interval: **30 minutes**

- Typically run every 30 minutes, may drift by up to 10 minutes.

Scheduled duration: **25 seconds**

- Periodic agents typically run for 25 seconds, may be terminated earlier.

Battery Saver mode can prevent execution

Per-device periodic agent limit: **6**

- User may be warned earlier

Resource Intensive agents run for a relatively long period of time when the phone meets a set of requirements relating to *processor* activity, *power* source, and *network* connection.

Example:

- Synchronizing large amounts of data when phone's not being used.

Resource Intensive Agent Constraints

- Duration: 10 minutes
- External power required
- Non-cellular connection required
- Minimum battery power: 90%
- Device screen lock required
- No active phone call
- Cannot change network to cellular

Resource Intensive Agent

Might never run on some devices.

Example: User doesn't have wi-fi access

But that's not all!

Additional limits for **both**:

- Unsupported API's
- Memory usage cap: 6MB
- Reschedule required: 2 weeks
- Unscheduled after 2 consecutive crashes

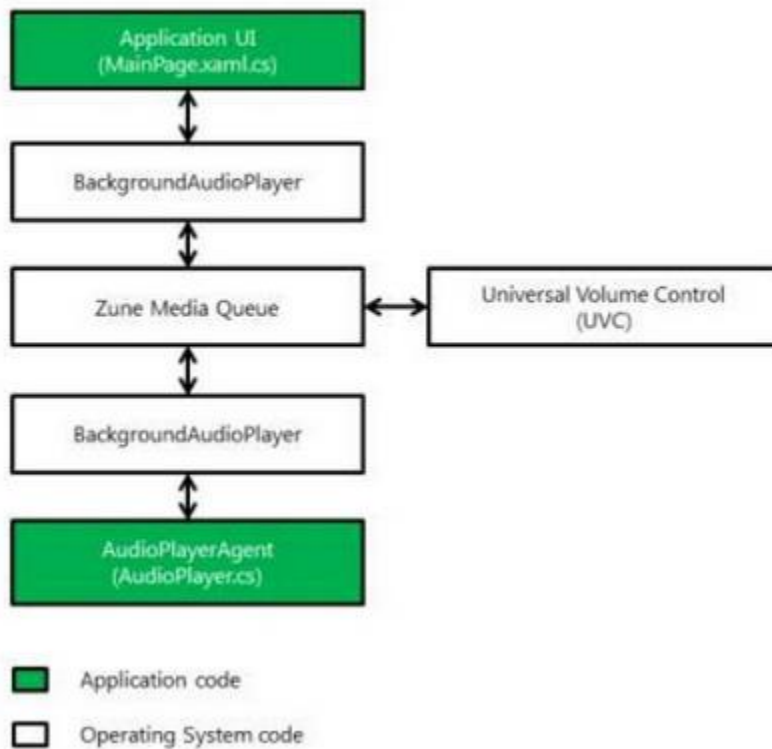
```
ShellToast toast = new ShellToast();  
toast.Title = "Background Agent Sample";  
toast.Content = toastMessage;  
toast.Show();  
  
#if DEBUG_AGENT  
    ScheduledActionService.LaunchForTest(  
        task.Name, TimeSpan.FromSeconds(60));  
#endif
```

Applications of Background Agents

Background Audio

- Continues to play when you navigate away
- Uses Zune playlist, Universal Volume Control
- Internet URI or Isolated Storage
- Implementation
 - BackgroundAudioPlayer
 - AudioPlayerAgent

Background Audio Playlist Application



Background File Transfer - Size

Maximum upload file size	5 MB
Maximum download size over cellular connection	20 MB – If a file exceeds this limit, the TransferPreferences property for the transfer is automatically changed to AllowBattery , which has the effect of requiring Wi-Fi for the transfer.
Maximum download size over Wi-Fi without external power	100 MB – Files larger than 100 MB must set the TransferPreferences property of the transfer to None or the transfer will fail. If you do not know the size of a transfer and it is possible that it could exceed this limit, you should set the value to None.

Background File Transfer - Limits

Maximum outstanding requests in the queue per application (this includes active and pending requests).	5 – Transfers are not removed from the queue automatically when they complete.
Maximum concurrent transfers across all applications on the device	2
Maximum queued transfers across all applications on the device	500

Alarm

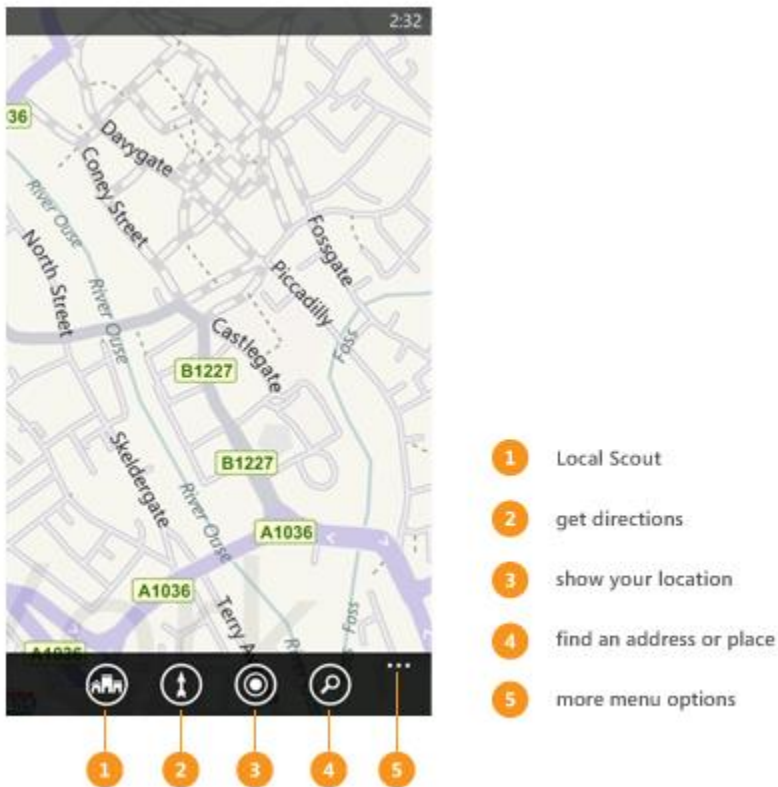
- Displays the Application Name
- **Always shows Alarm** as the UI title
- Displays text provided by the application
- Application can set alarm sound
- The sound begins quietly, gets louder
- Tapping Alarm takes user to initial page of app (same as first launch)

Reminder

- Displays a title that is provided by the app
- Uses the **default notification sound**
- When the user taps the Reminder, the **application can provide a navigation URI** and query string parameters to which the application will navigate when it is launched

Using maps and Locations

The Maps app in Windows Phone can show you where you are, where you want to go, and provide directions to get you there. It can also show you nearby shops or restaurants you might be interested in and what other people are saying about them.



Displaying a Map

To display a map in your Windows Phone 8 app, use the Map control. For more info, see How to add a Map control to a page in Windows Phone 8.

Important Note:

To use the control, you have to select the ID_CAP_MAP capability in the app manifest file. For more info, see How to modify the app manifest file for Windows Phone 8.

Displaying a Map with XAML

The following code example shows how you can use XAML to display a Map control in your Windows Phone 8 app.

```
<!--ContentPanel - place additional content here-->

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

    <maps:Map />

</Grid>
```

If you add the control by writing XAML, you also have to add the following xmlns declaration to the phone:PhoneApplicationPage element. If you drag and drop the Map control from the Toolbox, this declaration is added automatically.

```
xmlns:maps="clr-namespace:Microsoft.Phone.Maps.Controls;assembly=Microsoft.Phone.Maps"
```

Displaying a Map with code (C#)

The following code example shows how you can use code to display a Map control in your Windows Phone 8 app.

```
using Microsoft.Phone.Maps.Controls;
```

```
...
```

```
Map MyMap = new Map();
```

```
ContentPanel.Children.Add(MyMap);
```

Displaying a Map by using a built-in launcher

This topic describes how to write code that displays a map inside your app. If you simply want to display a map, you can also use the Maps task, which launches the built-in Maps app. For more info, see [How to use the Maps task for Windows Phone 8](#).

The following table lists all the built-in launchers that display or manage maps. For more info about launchers, see [Launchers and Choosers for Windows Phone 8](#).

Launcher	More info
Maps task	Launches the built-in Maps app and optionally marks a location.
Maps directions task	Launches the built-in Maps app and displays directions.
MapDownloader task	Downloads maps for offline use.

MapUpdater task	Checks for updates for offline maps that the user has previously downloaded.
------------------------	--

Specifying the center of a map (XAML)

You can set the center of the Map control by using its Center property. To set the property using XAML, assign a (latitude, longitude) pair to the Center property.

The following code example shows how you can set the center of Map by using XAML.

```
<!--ContentPanel - place additional content here-->

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

    <maps:Map x:Name="MyMap" Center="47.6097, -122.3331" />

</Grid>
```

The following code example shows how can set the **center of Map using code. (C#)**

using Microsoft.Phone.Maps.Controls;

using System.Device.Location;

...

```
Map MyMap = new Map();
```

```
MyMap.Center = new GeoCoordinate(47.6097, -122.3331);
```

```
ContentPanel.Children.Add(MyMap);
```

```
}
```

Specifying the zoom level of a map (XAML)

Use the ZoomLevel property to set the initial resolution at which you want to display the map. ZoomLevel property takes values from 1 to 20, where 1 corresponds to a fully zoomed out map, and higher zoom levels zoom in at a higher resolution. The following code examples show how you can set the zoom level of the map by using the ZoomLevel property in XAML and code.

The following code example shows how you can set the zoom level of the map by using the ZoomLevel property in XAML.

```
<!--ContentPanel - place additional content here-->  
  
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">  
    <maps:Map x:Name="MyMap" Center="47.6097, -122.3331" ZoomLevel="10"/>  
</Grid>
```

The following code example shows how you can set the zoom level of the map by using the ZoomLevel property in code. (C#)

```
using Microsoft.Phone.Maps.Controls;  
  
using System.Device.Location;  
  
...  
  
Map MyMap = new Map();  
  
MyMap.Center = new GeoCoordinate(47.6097, -122.3331);  
  
MyMap.ZoomLevel = 10;  
  
ContentPanel.Children.Add(MyMap);  
  
}
```

Converting a Geocoordinate to a GeoCoordinate

The Center property of the Map control requires a value of type GeoCoordinate from the System.Device.Location namespace. If you are using location services from the Windows.Devices.Geolocation namespace, you have to convert a Windows.Devices.Geolocation.Geocoordinate value to a System.Device.Location.GeoCoordinate value for use with the Map control.

You can get an extension method to do this conversion, along with other useful extensions to the Maps API, by downloading the Windows Phone Toolkit. If you want to write your own code, here is an example of a method that you can use to convert a Geocoordinate to a GeoCoordinate:

```
using System;  
  
using System.Device.Location; // Contains the GeoCoordinate class.  
  
using Windows.Devices.Geolocation; // Contains the Geocoordinate class.
```

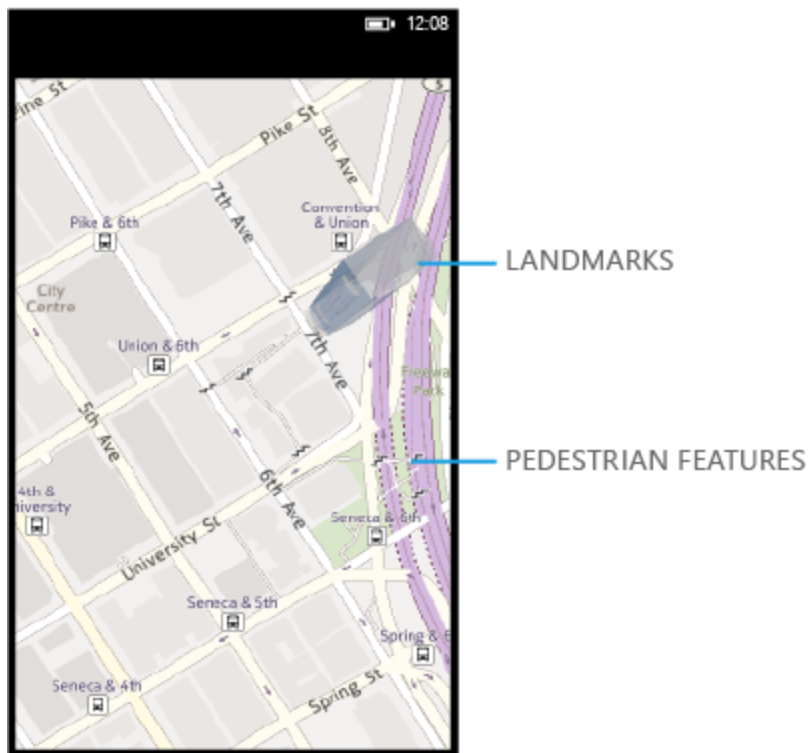
```
namespace CoordinateConverter
{
    public static class CoordinateConverter
    {
        public static GeoCoordinate ConvertGeocoordinate(Geocoordinate geocoordinate)
        {
            return new GeoCoordinate
            (
                geocoordinate.Latitude,
                geocoordinate.Longitude,
                geocoordinate.Altitude ?? Double.NaN,
                geocoordinate.Accuracy,
                geocoordinate.AltitudeAccuracy ?? Double.NaN,
                geocoordinate.Speed ?? Double.NaN,
                geocoordinate.Heading ?? Double.NaN
            );
        }
    }
}
```

Displaying landmarks and pedestrian features

Landmarks. Set the LandmarksEnabled property to true to display landmarks on a Map control. Landmarks are visible on the map only when the ZoomLevel property is set to a value of 16 or higher.

Pedestrian features. Set PedestrianFeaturesEnabled to true on a Map control to display pedestrian features such as public stairs. Pedestrian features are visible on the map only when the ZoomLevel property is set to a value of 16 or higher.

The following illustration displays a map with landmarks and pedestrian features.



The following example shows how you can set the `PedestrianFeaturesEnabled` property and the `LandmarksEnabled` property in XAML.

```
<!--ContentPanel - place additional content here-->
```

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
```

```
<maps:Map Center="47.6097, -122.3331" ZoomLevel="16" LandmarksEnabled="true"
PedestrianFeaturesEnabled="true"/>
```

```
</Grid>
```

The following example shows how to set these properties in code.

```
using Microsoft.Phone.Maps.Controls;
```

```
using System.Device.Location;
```

```
...
```

```
Map MyMap = new Map();
```

```
MyMap.Center = new GeoCoordinate(47.6097, -122.3331);
```

```
MyMap.ZoomLevel = 16;  
MyMap.LandmarksEnabled = true;  
MyMap.PedestrianFeaturesEnabled = true;  
ContentPanel.Children.Add(MyMap);  
}
```

Setting the cartographic mode

Once you set the center and zoom level of a map, you might also want to set the cartographic mode of the map. The cartographic mode defines the display and the translation of coordinate systems from screen coordinates to world coordinates on the Map control. You can use the `CartographicMode` property of the Map control to set the cartographic mode of the map. This property takes values from the `MapCartographicMode` enumeration. The following types of cartographic modes are supported in the `MapCartographicMode` enumeration:

Road: displays the normal, default 2-D map.

Aerial: displays an aerial photographic map.

Hybrid: displays an aerial view of the map overlaid with roads and labels.

Terrain: displays physical relief images for displaying elevation and water features such as mountains and rivers.

The following illustration displays the four cartographic modes.



The following example displays a map in the default Road mode. The buttons in the app bar can be used to view the map in Aerial, Hybrid, and Terrain modes.

XAML

```
<!--LayoutRoot is the root grid where all page content is placed-->
<Grid x:Name="LayoutRoot" Background="Transparent">

    <Grid.RowDefinitions>

        <RowDefinition Height="Auto"/>

        <RowDefinition Height="*/>

    </Grid.RowDefinitions>

    <!--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="Maps"      Style="{StaticResource
PhoneTextNormalStyle}"/>

        <TextBlock      x:Name="PageTitle"      Text="map      modes"      Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>

    </StackPanel>

    <!--ContentPanel - place additional content here-->

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

        <maps:Map x:Name="MyMap" Center="13.0810, 80.2740" ZoomLevel="10"/>

    </Grid>

</Grid>

<!--Sample code showing usage of ApplicationBar-->

<phone:PhoneApplicationPage.ApplicationBar>

    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">

        <shell:ApplicationBarIconButton IconUri="/Images/appbar_button1.png" Text="Road"
Click="Road_Click"/>

    </shell:ApplicationBar>

</phone:PhoneApplicationPage.ApplicationBar>
```

```
<shell:ApplicationBarIconButton IconUri="/Images/appbar_button2.png" Text="Aerial"
Click="Aerial_Click"/>
```

```
<shell:ApplicationBarIconButton IconUri="/Images/appbar_button3.png" Text="Hybrid"
Click="Hybrid_Click"/>
```

```
<shell:ApplicationBarIconButton IconUri="/Images/appbar_button4.png" Text="Terrain"
Click="Terrain_Click"/>
```

```
</shell:ApplicationBar>
```

```
</phone:PhoneApplicationPage.ApplicationBar>
```

C#

```
void Road_Click(object sender, EventArgs args)
```

```
{
```

```
    MyMap.CartographicMode = MapCartographicMode.Road;
```

```
}
```

```
void Aerial_Click(object sender, EventArgs args)
```

```
{
```

```
    MyMap.CartographicMode = MapCartographicMode.Aerial;
```

```
}
```

```
void Hybrid_Click(object sender, EventArgs args)
```

```
{
```

```
    MyMap.CartographicMode = MapCartographicMode.Hybrid;
```

```
}
```

```
void Terrain_Click(object sender, EventArgs args)
```

```
{
```

```
MyMap.CartographicMode = MapCartographicMode.Terrain;
}
```

Setting the color mode

You can display the map in a light color mode or a dark mode by using the `ColorMode` property. The values that this property can take—`Light` or `Dark`—is accepts are specified contained in the `MapColorMode` enumeration. The default is `Light`.

In the following illustration, the first map is in the `Light` color mode and the second map is in the `Dark` color mode.



The following code example displays a map in the default `Light` mode. The buttons in the app bar can be used to view the map in `Light` or `Dark` modes.

XAML

<!--LayoutRoot is the root grid where all page content is placed-->

```
<Grid x:Name="LayoutRoot" Background="Transparent">
```

```
<Grid.RowDefinitions>
```

```
<RowDefinition Height="Auto"/>
```

```
<RowDefinition Height="*" />
```

```

</Grid.RowDefinitions>

<!--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="Maps" Style="{StaticResource
PhoneTextNormalStyle}"/>

    <TextBlock x:Name="PageTitle" Text="color modes" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>

</StackPanel>

<!--ContentPanel - place additional content here-->

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

    <maps:Map x:Name="MyMap" />

</Grid>

</Grid>

<!--Sample code showing usage of ApplicationBar-->

<phone:PhoneApplicationPage.ApplicationBar>

    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">

        <shell:ApplicationBarIconButton IconUri="/Images/appbar_button1.png" Text="Light"
Click="Light_Click"/>

        <shell:ApplicationBarIconButton IconUri="/Images/appbar_button2.png" Text="Dark"
Click="Dark_Click"/>

    </shell:ApplicationBar>

</phone:PhoneApplicationPage.ApplicationBar>

```

C#

```

void Light_Click(object sender, EventArgs args)
{
    MyMap.ColorMode = MapColorMode.Light;
}

```

```
void Dark_Click(object sender, EventArgs args)
{
    MyMap.ColorMode = MapColorMode.Dark;
}
```

XAML

XAML --> Extensible Markup Language. XAML is very easy in use and it is tag based language. There are different tags that do their work. It is tag based and when we open a tag mostly it is necessary to close the same tag as same in HTML.

The Extensible Application Markup Language (XAML) with C# to create a simple "Hello, world" app that targets the Universal Windows Platform (UWP) on Windows 10. With a single project in Microsoft Visual Studio, you can build an app that runs on any Windows 10 device. Here we focus on creating an app that runs equally well on desktop and mobile devices.

Step 1: Create a new project in Visual Studio

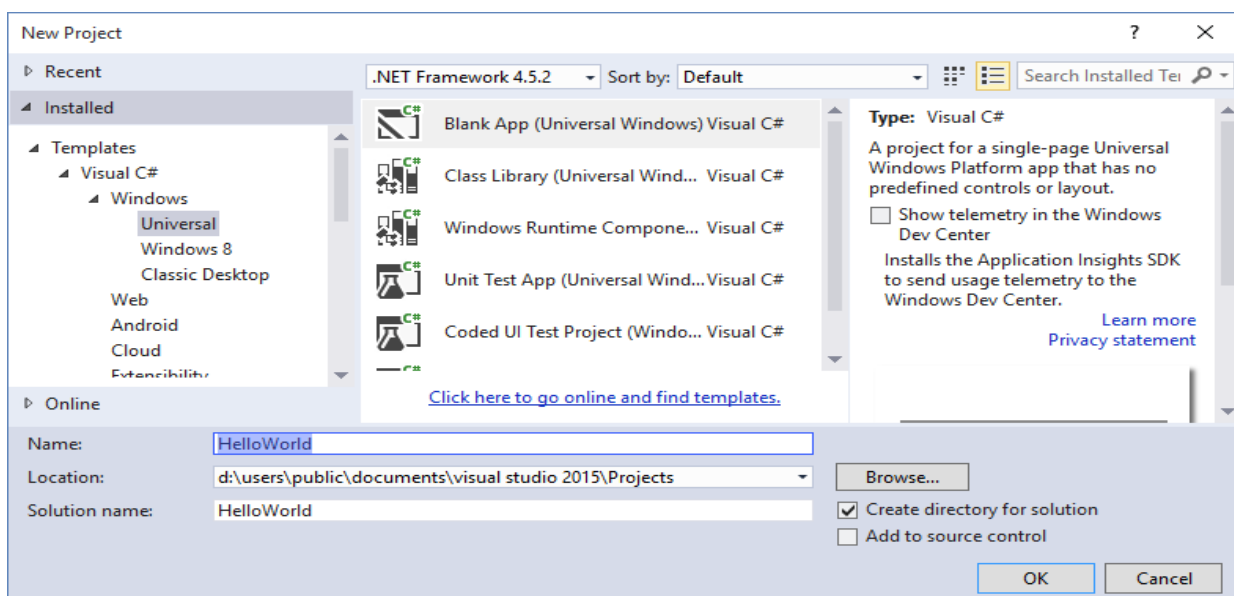
1. Launch Visual Studio 2015.

The Visual Studio 2015 Start page appears. (From now on, we'll refer to Visual Studio 2015 simply as Visual Studio .)

2. On the File menu, select New > Project.

The New Project dialog appears. The left pane of the dialog lets you select the type of templates to display.

3. In the left pane, expand Installed > Templates > Visual C# > Windows, then pick the Universal template group. The dialog's center pane displays a list of project templates for Universal Windows Platform (UWP) apps.

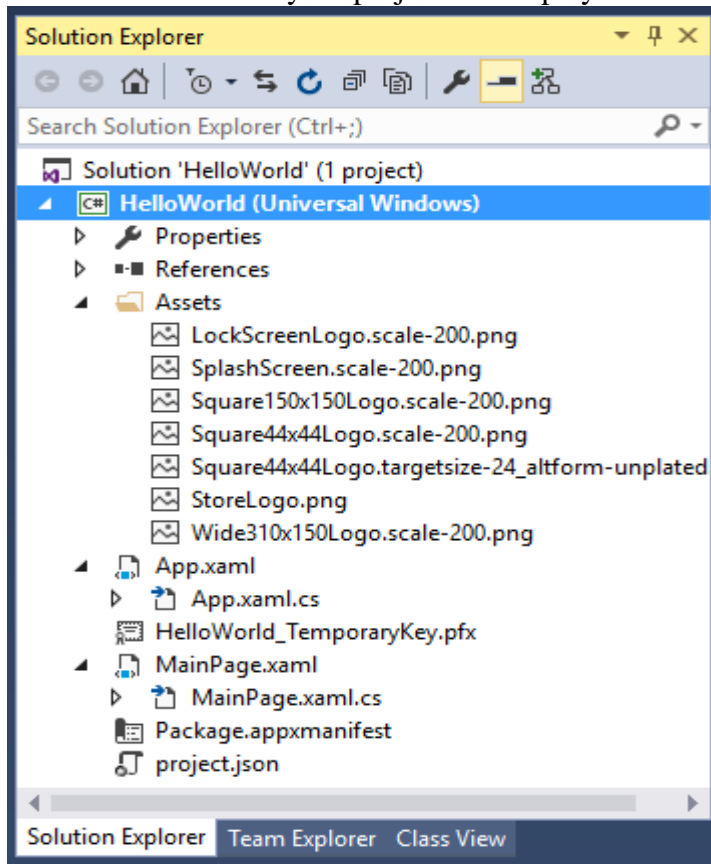


4. In the center pane, select the Blank App (Universal Windows) template.

The Blank App template creates a minimal UWP app that compiles and runs, but contains no user-interface controls or data. You add controls to the app over the course of this tutorial.

5. In the Name text box, type "HelloWorld".
6. Click OK to create the project.

Visual Studio creates your project and displays it in the Solution Explorer.



Although the Blank App is a minimal template, it still contains a lot of files:

- A manifest file (Package.appxmanifest) that describes your app (its name, description, tile, start page, and so on) and lists the files that your app contains.
- A set of logo images (Assets/Square150x150Logo.scale-200.png, Assets/Square44x44Logo.scale-200.png, and Assets/Wide310x150Logo.scale-200.png) to display in the start menu.
- An image (Assets/StoreLogo.png) to represent your app in the Windows Store.
- A splash screen (Assets/SplashScreen.scale-200.png) to display when your app starts.
- XAML and code files for the app (App.xaml and App.xaml.cs).

- A start page (MainPage.xaml) and an accompanying code file (MainPage.xaml.cs) that run when your app starts.

These files are essential to all UWP apps using C#. Every project that you create in Visual Studio contains them.

Step 2: Modify your start page

What's in the files?

To view and edit a file in your project, double-click the file in the Solution Explorer. By default, you can expand a XAML file just like a folder to see its associated code file. XAML files open in a split view that shows both the design surface and the XAML editor.

In this tutorial, you work with just a few of the files listed previously: App.xaml, MainPage.xaml, and MainPage.xaml.cs.

App.xaml and App.xaml.cs

App.xaml is where you declare resources that are used across the app. App.xaml.cs is the code-behind file for App.xaml. Code-behind is the code that is joined with the XAML page's partial class. Together, the XAML and code-behind make a complete class. App.xaml.cs is the entry point for your app. Like all code-behind pages, it contains a constructor that calls the InitializeComponent method. You don't write the InitializeComponent method. It's generated by Visual Studio, and its main purpose is to initialize the elements declared in the XAML file. App.xaml.cs also contains methods to handle activation and suspension of the app.

MainPage.xaml

In MainPage.xaml you define the UI for your app. You can add elements directly using XAML markup, or you can use the design tools provided by Visual Studio. MainPage.xaml.cs is the code-behind page for MainPage.xaml. It's where you add your app logic and event handlers. Together these two files define a new class called MainPage, which inherits from Page, in the HelloWorld namespace.

MainPage.xaml

```
<Page
  x:Class="HelloWorld.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:HelloWorld"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d">

  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
```

```

    </Grid>
</Page>
MainPage.xaml.cs

```

```

using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace HelloWorld
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }
    }
}

```

Modify the start page

Now, let's add some content to the app. **To modify the start page**

1. Double-click MainPage.xaml in **Solution Explorer** to open it.
2. In the XAML editor, add the controls for the UI.

In the root **Grid**, add this XAML. It contains a **StackPanel** with a title **TextBlock**, a **TextBlock** that asks the user's name, a **TextBox** element to accept the user's name, a **Button**, and another **TextBlock** to show a greeting. Some of these controls have names so that you can refer to them later in your code.

```

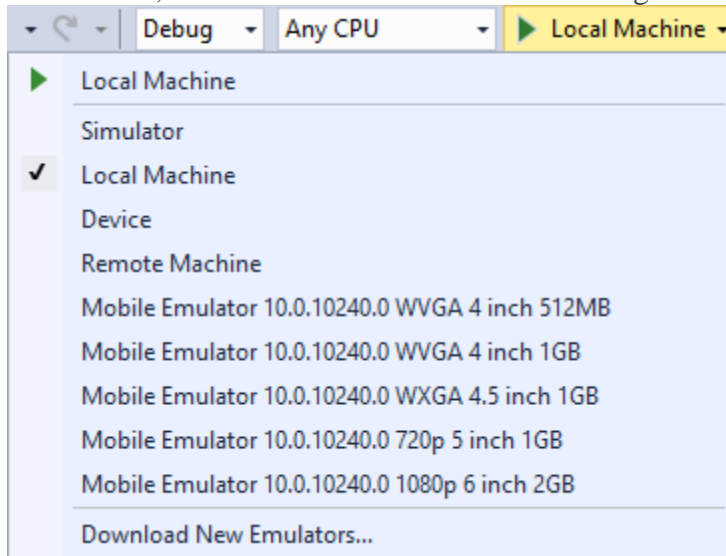
<StackPanel x:Name="contentPanel" Margin="8,32,0,0">
    <TextBlock Text="Hello, world!" Margin="0,0,0,40"/>
    <TextBlock Text="What's your name?"/>
    <StackPanel x:Name="inputPanel" Orientation="Horizontal" Margin="0,20,0,20">
        <TextBox x:Name="nameInput" Width="280" HorizontalAlignment="Left"/>
        <Button x:Name="inputButton" Content="Say "Hello""/>
    </StackPanel>
    <TextBlock x:Name="greetingOutput"/>
</StackPanel>

```

The controls that you added in the XAML editor show up in the design view.

Step 3: Start the app

At this point, you've created a very simple app. This is a good time to build, deploy, and launch your app and see what it looks like. You can debug your app on the local machine, in a simulator or emulator, or on a remote device. Here's the target device menu in Visual Studio.

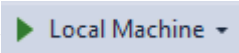



Start the app on a Desktop device

By default, the app runs on the local machine. The target device menu provides several options for debugging your app on devices from the desktop device family.

- **Simulator**
- **Local Machine**
- **Remote Machine**

To start debugging on the local machine

1. In the target device menu () on the **Standard** toolbar, make sure that **Local Machine** is selected. (It's the default selection.)
2. Click the **Start Debugging** button () on the toolbar.

—or—

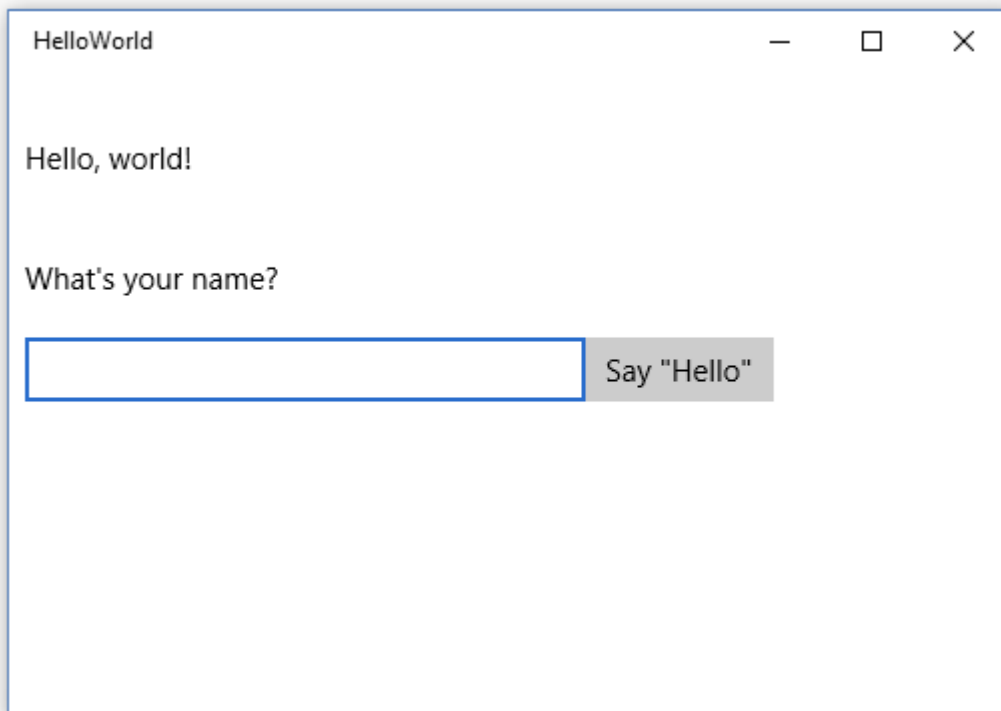
From the **Debug** menu, click **Start Debugging**.

—or—

Press F5.

The app opens in a window, and a default splash screen appears first. The splash screen is defined by an image (SplashScreen.png) and a background color (specified in your app's manifest file).


The splash screen disappears, and then your app appears. It looks like this.



Press the Windows key to open the **Start** menu, then show all apps. Notice that deploying the app locally adds its tile to the **Start** menu. To run the app again (not in debugging mode), tap or click its tile in the **Start** menu.

It doesn't do much—yet—but congratulations, you've built your first UWP app!

To stop debugging

- Click the **Stop Debugging** button () in the toolbar.
—or—
From the **Debug** menu, click **Stop debugging**.
—or—
Close the app window.

Start the app on a mobile device emulator



Your app runs on any Windows 10 device, so let's see how it looks on a Windows Phone.

In addition to the options to debug on a desktop device, Visual Studio provides options for deploying and debugging your app on a physical mobile device connected to the computer, or on a mobile device emulator. You can choose among emulators for devices with different memory and display configurations.

- **Device**
- **Emulator <SDK version> WVGA 4 inch 512MB**
- **Emulator <SDK version> WVGA 4 inch 1GB**
- etc... (Various emulators in other configurations)

It's a good idea to test your app on a device with a small screen and limited memory, so use the **Emulator 10.0.10240.0 WVGA 4 inch 512MB** option.

To start debugging on a mobile device emulator

1. In the target device menu () on the **Standard** toolbar, pick **Emulator 10.0.10240.0 WVGA 4 inch 512MB**.
2. Click the **Start Debugging** button () in the toolbar.

–or–

From the **Debug** menu, click **Start Debugging**.

–or–

Press F5.

Visual Studio starts the selected emulator and then deploys and starts your app. On the mobile device emulator, the app looks like this.



The first thing you'll notice is the button is pushed off the smaller screen of a mobile device. Later in this tutorial, you'll learn how to adapt the UI to different screen sizes so your app always looks good.

You might also notice that you can type in the **TextBox**, but right now, clicking or tapping the **Button** doesn't do anything. In the next steps, you create an event handler for the button's **Click** event to display a personalized greeting. You add the event handler code to your MainPage.xaml.cs file.

Step 4: Create an event handler


XAML elements can send messages when certain events occur. These event messages give you the opportunity to take some action in response to the event. You put your code to respond to the event in an event handler method. One of the most common events in many apps is a user clicking a **Button**.

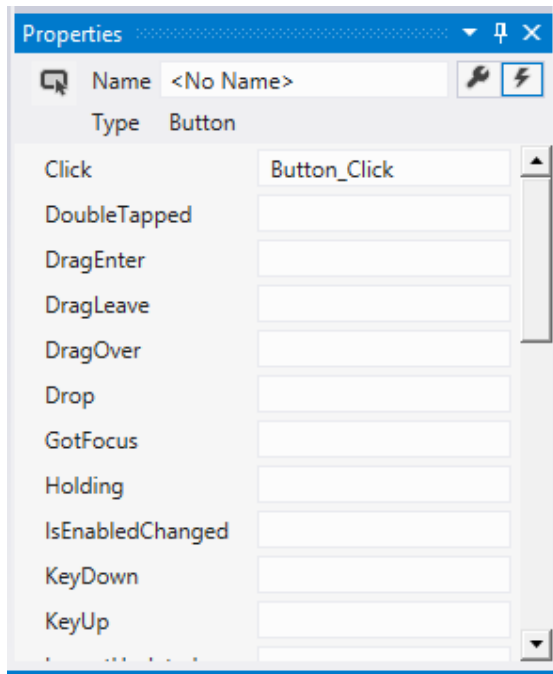
Let's create an event handler for your button's **Click** event. The event handler will get the user's name from the nameInput**TextBox** control and use it to output a greeting to the greetingOutput **TextBlock**.

Using events that work for touch, mouse, and pen input

What events should you handle? Because they can run on a variety of devices, design your Windows Store apps with touch input in mind. Your app must also be able to handle input from a mouse or a stylus. Fortunately, events such as **Click** and **DoubleTapped** are device-independent. If you're familiar with Microsoft .NET programming, you might have seen separate events for mouse, touch, and stylus input, like **MouseMove**, **TouchMove**, and **StylusMove**. In Windows Store apps, these separate events are replaced with a single **PointerMoved** event that works equally well for touch, mouse, and stylus input.

To add an event handler

1. In XAML or design view, select the "Say Hello" **Button** that you added to MainPage.xaml.
2. In the **Properties Window**, click the Events button (.
3. Find the **Click** event at the top of the event list. In the text box for the event, type the name of the function that handles the **Click** event. For this example, type "Button_Click".



4. Press Enter. The event handler method is created and opened in the code editor so you can add code to be executed when the event occurs.

In the XAML editor, the XAML for the **Button** is updated to declare the **Click** event handler like this.

```
<Button x:Name="inputButton" Content="Say &quot;Hello&quot;" Click="Button_Click"/>
```

5. Add code to the event handler that you created in the code-behind page. In the event handler, retrieve the user's name from the nameInput **TextBox** control and use it to create a greeting. Use the greetingOutput **TextBlock** to display the result.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    greetingOutput.Text = "Hello, " + nameInput.Text + "!";
}
```

6. Debug the app on the local machine. When you enter your name in the text box and click the button, the app displays a personalized greeting.

Step 5: Adapt the UI to different window sizes

Now we'll make the UI adapt to different screen sizes so it looks good on mobile devices. To do this, you add a **VisualStateManager** and set properties that are applied for different visual states.

To adjust the UI layout

1. In the XAML editor, add this block of XAML after the opening tag of the root **Grid** element.

```
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup>
    <VisualState x:Name="wideState">
      <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="641" />
      </VisualState.StateTriggers>
    </VisualState>
    <VisualState x:Name="narrowState">
      <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="0" />
      </VisualState.StateTriggers>
      <VisualState.Setters>
        <Setter Target="inputPanel.Orientation" Value="Vertical"/>
        <Setter Target="inputButton.Margin" Value="0,4,0,0"/>
      </VisualState.Setters>
    </VisualState>
  </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

2. Debug the app on the local machine. Notice that the UI looks the same as before unless the window gets narrower than 641 pixels.
3. Debug the app on the mobile device emulator. Notice that the UI uses the properties you defined in the narrowState and appears correctly on the small screen.



If you've used a **VisualStateManager** in previous versions of XAML, you might notice that the XAML here uses a simplified syntax.

The **VisualState** named **wideState** has an **AdaptiveTrigger** with its **MinWindowWidth** property set to 641. This means that the state is to be applied only when the window width is not less than the minimum of 641 pixels. You don't define any **Setter** objects for this state, so it uses the layout properties you defined in the XAML for the page content.

The second **VisualState**, **narrowState**, has an **AdaptiveTrigger** with its **MinWindowWidth** property set to 0. This state is applied when the window width is greater than 0, but less than 641 pixels. (At 641 pixels, the **wideState** is applied.) In this state, you do define some **Setter** objects to change the layout properties of controls in the UI:

- You change the **Orientation** of the **inputPanel** element from **Horizontal** to **Vertical**.
- You add a top margin of 4 to the **inputButton** element.

Summary

Congratulations, you've created your first app for Windows 10 and the UWP!

Adding controls and handling events (XAML)

You create the UI for your app by using controls such as buttons, text boxes, and combo boxes. Here we show you how to add controls to your app. You typically use this pattern when working with controls:

- You add a control to your app UI.
- You set properties on the control, such as width, height, or foreground color.
- You hook up some code to the control so that it does something.

Adding a control

You can add a control to an app in several ways:

- Use a design tool like Blend for Visual Studio or the Microsoft Visual Studio XAML designer.
- Add the control to the XAML markup in the Visual Studio XAML editor.
- Add the control in code. Controls that you add in code are visible when the app runs, but are not visible in the Visual Studio XAML designer.

Documentation for each control includes a "How to" topic that describes how to add the control in XAML, code, or using a design tool. For example, to add a **Button** control, see [How to add a button](#).

Here, we use Visual Studio as our design tool, but you can do the same tasks and more in Blend for Visual Studio.

In Visual Studio, when you add and manipulate controls in your app, you can use many of the program's features, including the **Toolbox**, XAML designer, XAML editor, and the **Properties** window.

The Visual Studio **Toolbox** displays many of the controls that you can use in your app. To add a control to your app, double-click it in the **Toolbox**. For example, when you double-click the **TextBox** control, this XAML is added to the **XAML**view.

```
<TextBox HorizontalAlignment="Left" Text="TextBox" VerticalAlignment="Top"/>
```

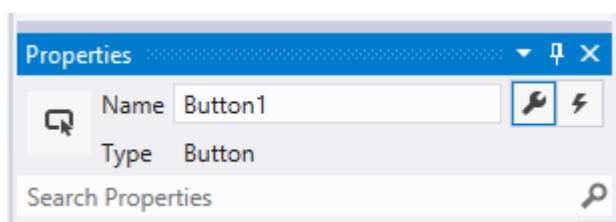
You can also drag the control from the **Toolbox** to the XAML designer.

Setting the name of a control

To work with a control in code, you set its [x:Name](#) attribute and reference it by name in your code. You can set the name in the Visual Studio **Properties** window or in XAML. Here's how to change the name of the currently selected control by using the **Name** text box at the top of the **Properties** window.

To name a control

1. Select the element to name.
2. In the **Properties** panel, type a name into the **Name** text box.
3. Press Enter to commit the name.

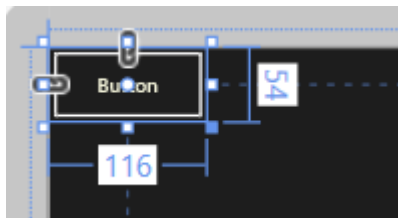


Here's how you can change the name of a control in the XAML editor by changing the [x:Name](#) attribute.

```
<Button x:Name="Button1" Content="Button"/>
```

Setting control properties

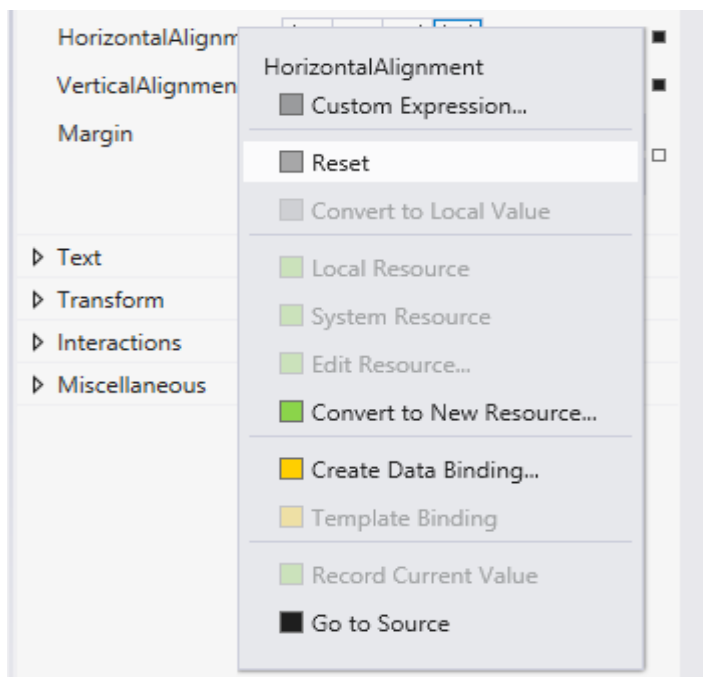
You use properties to specify the appearance, content, and other attributes of controls. When you add a control using a design tool, some properties that control size, position, and content might be set for you by Visual Studio. You can change some properties, such as **Width**, **Height** or **Margin**, by selecting and manipulating the control in the **Design** view. This illustration shows some of the resizing tools available in **Design** view.



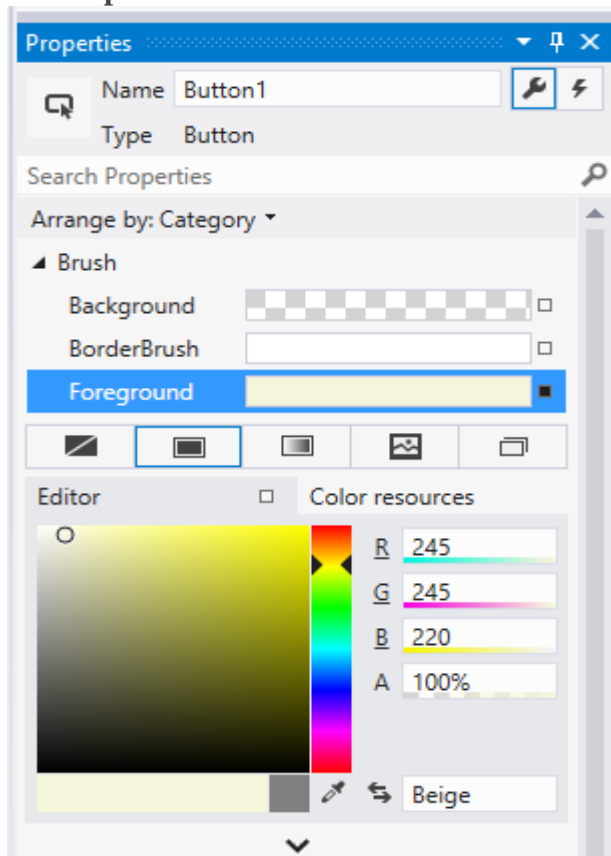
You might want to let the control be sized and positioned automatically. In this case, you can reset the size and position properties that Visual Studio set for you.

To reset a property

1. In the **Properties** panel, click the property marker next to the property value. The property menu opens.
2. In the property menu, click **Reset**.



You can set control properties in the **Properties** window, in XAML, or in code. For example, to change the foreground color for a **Button**, you set the control's **Foreground** property. This illustration shows how to set the **Foreground** property by using the color picker in the **Properties** window.

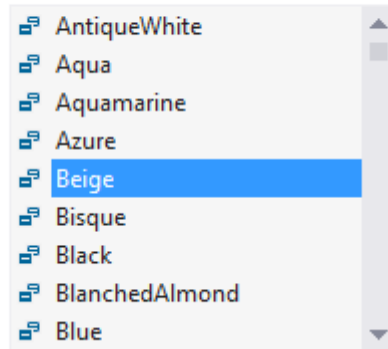


Here's how to set the **Foreground** property in the **XAML** editor. Notice the Visual Studio IntelliSense window that opens to help you with the syntax.

```
<Button x:Name="Button1" Content="Button"
        HorizontalAlignment="Left" VerticalAlignment="Top"
        For/>
```



```
<Button x:Name="Button1" Content="Button"
        HorizontalAlignment="Left" VerticalAlignment="Top"
        Foreground="b"/>
```



Here's the resulting XAML after you set the **Foreground** property.
XAML

```
<Button x:Name="Button1" Content="Button"
        HorizontalAlignment="Left" VerticalAlignment="Top"
        Foreground="Beige"/>
```

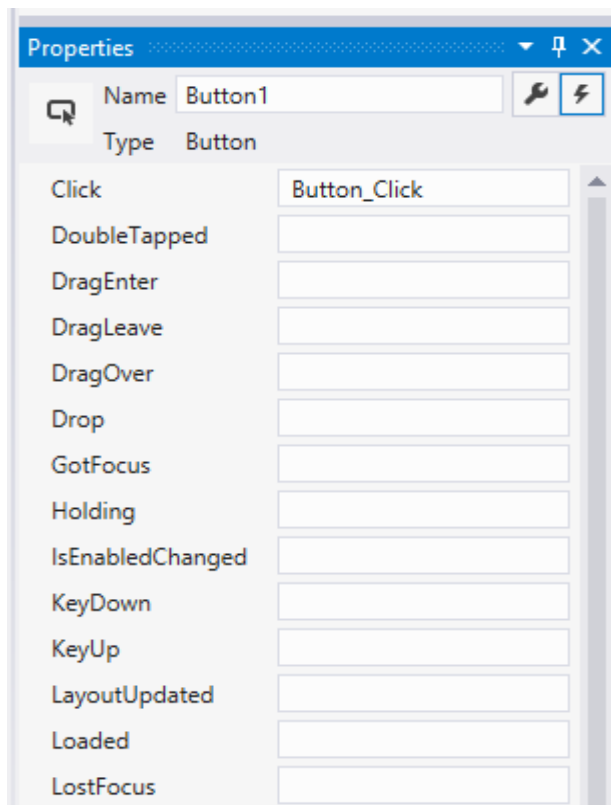
Here's how to set the **Foreground** property in code. C#, [C++](#), [VB](#)

```
Button1.Foreground = new SolidColorBrush(Windows.UI.Colors.Beige);
```

Creating an event handler

Each control has events that enable you to respond to actions from your user or other changes in your app. For example, a **Button** control has a **Click** event that is raised when a user clicks the **Button**. You create a method, called an event handler, to handle the event. You can associate a control's event with an event handler method in the **Properties** window, in XAML, or in code. For more info about events, see [Events and routed events overview](#).

To create an event handler, select the control and then click the **Events** tab at the top of the **Properties** window. The **Properties** window lists all of the events available for that control. Here are some of the events for a **Button**.



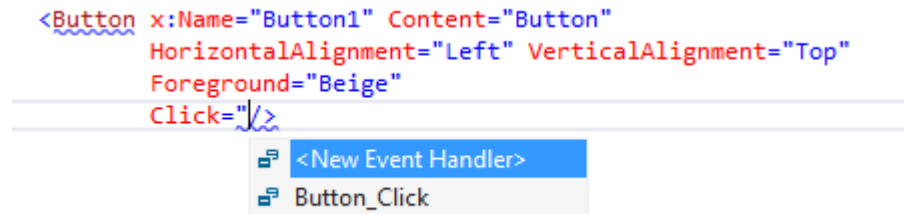
To create an event handler with the default name, double-click the text box next to the event name in the **Properties** window. To create an event handler with a custom name, type the name of your choice into the text box and press enter. The event handler is created and the code-behind file is opened in the code editor. The event handler method has 2 parameters. The first is sender, which is a reference to the object where the handler is attached. The sender parameter is an **Object** type. You typically cast sender to a more precise type if you expect to check or change state on the sender object itself. Based on your own app design, you expect a type that is safe to cast sender to, based on where the handler is attached. The second value is event data, which generally appears in signatures as the e parameter.

Here's code that handles the **Click** event of a **Button** named Button1. When you click the button, the **Foreground** property of the **Button** you clicked is set to blue. C#, [C++](#), [VB](#)

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    Button b = (Button)sender;
    b.Foreground = new SolidColorBrush(Windows.UI.Colors.Blue);
}
```

You can also associate an event handler in XAML. In the XAML editor, you type in the event name that you want to handle. Visual Studio shows an IntelliSense window when you begin typing. After you specify the event, you can double-click <New Event Handler> in the IntelliSense window to create a new event handler with the default name, or select an existing

event handler from the list. Here's the IntelliSense window that appears to help you create a new event handler.



This example shows how to associate a **Click** event with an event handler named Button_Click in XAML.

```
<Button Name="Button1" Content="Button" Click="Button_Click"/>
```

You can also associate an event with its event handler in the code-behind. Here's how to associate an event handler in code. C#, C++, VB

```
Button1.Click += new RoutedEventArgs(Button_Click);
```

New controls

If you use other XAML platforms, you might be interested in these controls that are new for Windows 8.

- **AppBar**
- **CaptureElement**
- **FlipView**
- **GridView**
- **SemanticZoom**
- **ProgressRing**
- **ToggleSwitch**
- **VariableSizedWrapGrid**

Displaying text (XAML)

The XAML framework provides several controls for rendering text, and a set of properties for formatting the text. The controls for displaying read-only text are **TextBlock** and **RichTextBlock**. This quickstart shows you how to use **TextBlock** controls to display text.

TextBlock

TextBlock is the primary control for displaying read-only text in Windows Runtime apps using C++, C#, or Visual Basic. You can display text in a **TextBlock** control using its **Text** property. This XAML shows how to define a **TextBlock** control and set its **Text** property to a string.

XAML

```
<TextBlock Text="Hello, world!" />
```

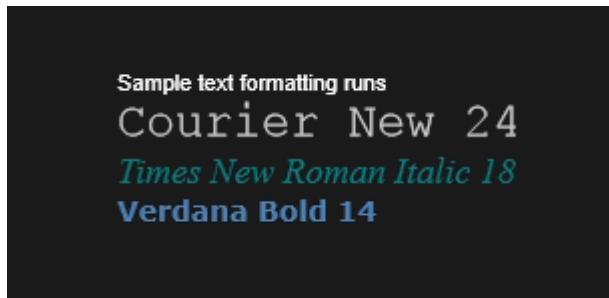
You can also display a series of strings in a **TextBlock**, where each string has different formatting. You can do this by using a **Run** element to display each string with its formatting and by separating each **Run** element with a **LineBreak** element.

Here's how to define several differently formatted text strings in a **TextBlock** by using **Run** objects separated with a **LineBreak**.

XAML

```
<TextBlock FontFamily="Arial" Width="400" Text="Sample text formatting runs">
  <LineBreak/>
  <Run Foreground="LightGray" FontFamily="Courier New" FontSize="24">
    Courier New 24
  </Run>
  <LineBreak/>
  <Run Foreground="Teal" FontFamily="Times New Roman" FontSize="18"
  FontStyle="Italic">
    Times New Roman Italic 18
  </Run>
  <LineBreak/>
  <Run Foreground="SteelBlue" FontFamily="Verdana" FontSize="14" FontWeight="Bold">
    Verdana Bold 14
  </Run>
</TextBlock>
```

Here's the result.



Summary and next steps

You learned how to create **TextBlock** controls to display text in your app.

Adding text input and editing controls (XAML)

The XAML framework includes several controls for entering and editing text, and a set of properties for formatting the text. The text-entry controls are **TextBox**, **PasswordBox**, and **RichEditBox**. This quickstart shows you how you can use these text controls to display, enter, and edit text.

Choosing a text control

The XAML framework includes 3 core text-entry controls: **TextBox**, **PasswordBox**, and **RichEditBox**. The text control that you use depends on your scenario. Here are some scenarios and the recommended control.

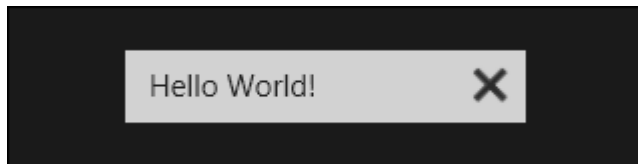
Scenario	Recommended Control
Enter or edit plain text, such as in a form.	TextBox
Enter a password.	PasswordBox
Edit a document, article, or blog that requires formatting, paragraphs, hyperlinks, or inline images.	RichEditBox

TextBox

You can use a **TextBox** control to enter and edit unformatted text. You can use the **Text** property to get and set the text in a **TextBox**. Here's the XAML for a simple **TextBox** with its **Text** property set.

```
<TextBox Height="35" Width="200" Text="Hello World!" Margin="20"/>
```

Here's the **TextBox** that results from this XAML.



You can make a **TextBox** read-only by setting the **IsReadOnly** property to **true**. To make the text in a multi-line **TextBox** wrap, set the **TextWrapping** property to **Wrap** and the **AcceptsReturn** property to **true**.

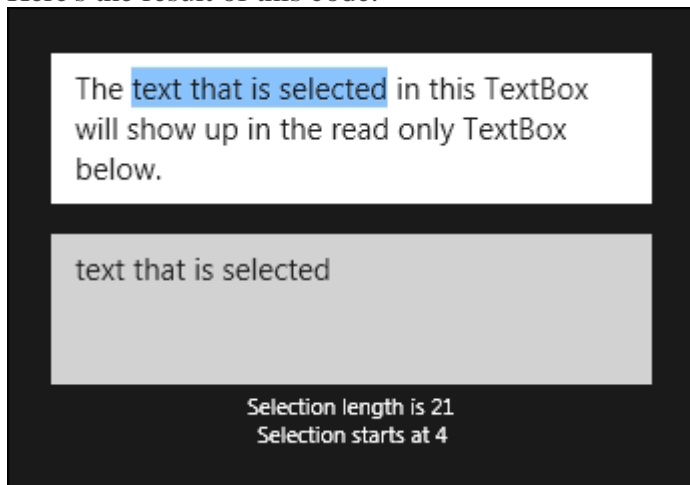
You can get or set the selected text in a **TextBox** using the **SelectedText** property. Use the **SelectionChanged** event to do something when the user selects or de-selects text.

Here, we have an example of these properties and methods in use. When you select text in the first **TextBox**, the selected text is displayed in the second **TextBox**, which is read-only. The values of the **SelectionLength** and **SelectionStart** properties are shown in two **TextBlocks**. This is done using the **SelectionChanged** event.

```
<TextBox x:Name="textBox1" Height="75" Width="300" Margin="10"
    Text="The text that is selected in this TextBox will show up in the read only TextBox below."
    TextWrapping="Wrap" AcceptsReturn="True"
    SelectionChanged="TextBox1_SelectionChanged" />
<TextBox x:Name="textBox2" Height="75" Width="300" Margin="5"
    TextWrapping="Wrap" AcceptsReturn="True" IsReadOnly="True"/>
<TextBlock x:Name="label1" HorizontalAlignment="Center"/>
<TextBlock x:Name="label2" HorizontalAlignment="Center"/>
```

```
private void TextBox1_SelectionChanged(object sender, RoutedEventArgs e)
{
    textBox2.Text = textBox1.SelectedText;
    label1.Text = "Selection length is " + textBox1.SelectionLength.ToString();
    label2.Text = "Selection starts at " + textBox1.SelectionStart.ToString();
}
```

Here's the result of this code.



PasswordBox

You can enter a single line of non-wrapping content in a **PasswordBox** control. The user cannot view the entered text; only password characters that represents the text are displayed. You can specify this password character by using the **PasswordChar** property, and you can specify the maximum number of characters that the user can enter by setting the **MaxLength** property.

You get the text that the user entered from the **Password** property, typically in the handler for the **PasswordChanged** event.

Here's the XAML for a password box control that demonstrates the default look of the **PasswordBox**. When the user enters a password, it is checked to see if it is the literal value, "Password". If it is, we display a message to the user.

XAML

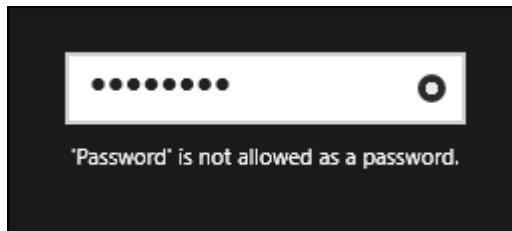
```
<PasswordBox x:Name="pwBox" Height="35" Width="200"
    MaxLength="8" PasswordChanged="pwBox_PasswordChanged"/>

<TextBlock x:Name="statusText" Margin="10" HorizontalAlignment="Center" />
```

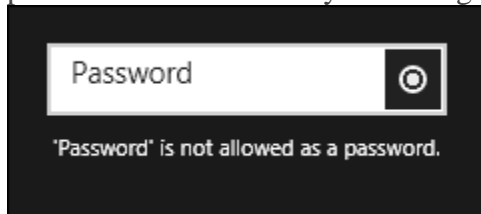
// C#

```
private void pwBox_PasswordChanged(object sender, RoutedEventArgs e)
{
    if (pwBox.Password == "Password")
    {
        statusText.Text = "'Password' is not allowed as a password.";
    }
}
```

Here's the result when this code runs and the user enters "Password".



In Windows Store apps, the **PasswordBox** has a built-in button that the user can touch or click to display the password text. Here's the result of the user's action. When the user releases it, the password is automatically hidden again.



In Windows Phone Store apps, the **PasswordBox** has a built-in checkbox below it that the user can check to display the password text.



RichEditBox

You can use a **RichEditBox** control to enter and edit rich text documents that contain formatted text, hyperlinks, and images. You can make a **RichEditBox** read-only by setting its **IsReadOnly** property to **true**.

By default, the **RichEditBox** supports spell checking. To disable the spell checker, set the **IsSpellCheckEnabled** property to false. For more info, see [Guidelines and checklist for spell checking](#).

You use the **Document** property of the **RichEditBox** to get its content. The content of a **RichEditBox** is a **Windows.UI.Text.ITextDocument** object, unlike the **RichTextBlock** control, which uses **Windows.UI.Xaml.Documents.Block** objects as its content. The **ITextDocument** interface provides a way to load and save the document to a stream, retrieve text ranges, get the active selection, undo and redo changes, set default formatting attributes, and so on.

This example shows how to load and save a Rich Text Format (rtf) file in a **RichEditBox**.

```
<Grid Margin="120">
    <Grid.RowDefinitions>
        <RowDefinition Height="50"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <StackPanel Orientation="Horizontal">
        <Button Content="Open file" Click="OpenButton_Click"/>
        <Button Content="Save file" Click="SaveButton_Click"/>
    </StackPanel>

    <RichEditBox x:Name="editor" Grid.Row="1"/>
</Grid>
```

```
private async void OpenButton_Click(object sender, RoutedEventArgs e)
{
    // Open a text file.
    Windows.Storage.Pickers.FileOpenPicker open =
        new Windows.Storage.Pickers.FileOpenPicker();
    open.SuggestedStartLocation =
        Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
```

```
open.FileTypeFilter.Add(".rtf");

Windows.Storage.StorageFile file = await open.PickSingleFileAsync();

if (file != null)
{
    Windows.Storage.Streams.IRandomAccessStream randAccStream =
        await file.OpenAsync(Windows.Storage.FileAccessMode.Read);

    // Load the file into the Document property of the RichEditBox.
    editor.Document.LoadFromStream(Windows.UI.Text.TextSetOptions.FormatRtf,
randAccStream);
}
}

private async void SaveButton_Click(object sender, RoutedEventArgs e)
{
    if (((ApplicationView.Value != ApplicationViewState.Snapped) ||
        ApplicationView.TryUnsnap()))
    {
        FileSavePicker savePicker = new FileSavePicker();
        savePicker.SuggestedStartLocation = PickerLocationId.DocumentsLibrary;

        // Dropdown of file types the user can save the file as
        savePicker.FileTypeChoices.Add("Rich Text", new List<string>() { ".rtf" });

        // Default file name if the user does not type one in or select a file to replace
        savePicker.SuggestedFileName = "New Document";

        StorageFile file = await savePicker.PickSaveFileAsync();
        if (file != null)
        {
            // Prevent updates to the remote version of the file until we
            // finish making changes and call CompleteUpdatesAsync.
            CachedFileManager.DeferUpdates(file);
            // write to file
            Windows.Storage.Streams.IRandomAccessStream randAccStream =
                await file.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);

            editor.Document.SaveToStream(Windows.UI.Text.TextGetOptions.FormatRtf,
randAccStream);

            // Let Windows know that we're finished changing the file so the
```

```
// other app can update the remote version of the file.
FileUpdateStatus status = await CachedFileManager.CompleteUpdatesAsync(file);
if (status != FileUpdateStatus.Complete)
{
    Windows.UI.Popups.MessageDialog errorBox =
        new Windows.UI.Popups.MessageDialog("File " + file.Name + " couldn't be
saved.");
    await errorBox.ShowAsync();
}
}
```

Using the touch keyboard

The touch keyboard can be used for text entry when your app runs on a device with a touch screen. The touch keyboard is invoked when the user taps on an editable input field, such as a **TextBox** or **PasswordBox**, and is dismissed when the input field loses focus. The touch keyboard uses accessibility info to determine when it is invoked and dismissed. The text controls provided in the XAML framework have the automation properties built in. If you create your own custom text controls, you must implement **TextPattern** to use the touch keyboard.

Summary

You learned how to create **TextBox**, **PasswordBox**, and **RichEditBox** controls to display and edit text in your app.

Packaging Universal Windows apps

To sell your Universal Windows app or distribute it to other users, you need to create an appxupload package for it. When you create the appxupload, another appx package will be generated to use for testing and sideloading. You can distribute your app directly by sideloading the appx package to a device.

For Windows 10, you generate one package (.appxupload) that can be uploaded to the Windows Store. Your app is then available to be installed and run on any Windows 10 device.

Here are the steps:

1. [Before packaging your app](#): Follow these steps to make sure your application is ready to be packaged for store submission.
2. [Configure an app package](#): Use the manifest designer to configure the package. For example, add tile images and choose the orientations that your app supports.
3. [Create an app package](#): Use the wizard in Visual Studio and then certify your package with the Windows App Certification Kit.
4. [Sideload your app package](#): After sideloading your app to a device, you can test it works correctly.

Once you've done this, you are ready to sell your app in the Store. If you have a line-of-business (LOB) app, that you don't plan to sell because it is for internal users only, you can sideload this app to install it on any Windows 10 device.

[Before packaging your app](#)

1. **Test your app**: Before you package your app for store submission, make sure it works as expected on all device families that you plan to support. These device families may include desktop, mobile, Surface Hub, XBOX, IoT devices, or others.
2. **Optimize your app**: You can use Visual Studio's profiling and debugging tools to optimize the performance of your Universal Windows app. For example, the Timeline tool for UI responsiveness, the memory Usage tool, the CPU Usage tool, and more.
3. **Check .NET Native compatibility (for VB and C# apps)**: With the Universal Windows Platform, there is now a new native compiler that will improve the runtime performance of your app. With this change, it is highly recommended that you test your app in this compilation environment. By default, the **Release** build configuration enables the .NET native toolchain, so it is important to test your app with this **Release** configuration and check that your app behaves as expected.

Configure an app package

The app manifest file (package.appxmanifest.xml) has the properties and settings that are required to create your app package. For example, properties in the manifest file describe the image to use as the tile of your app and the orientations that your app supports when a user rotates the device.

Visual Studio has a manifest designer that makes it easy for you to update the manifest file without editing the raw XML of the file.





Visual Studio can associate your package with the Store. When you do this, some of the fields in the **Packaging** tab of the manifest designer are automatically updated.

Configure a package with the manifest designer

1. In **Solution Explorer**, expand the project node of your Universal Windows app.
2. Double-click the **Package.appxmanifest** file.
If the manifest file is already open in XML code view, Visual Studio prompts you to close the file.
3. Now you can decide how to configure your app. Each tab contains information that you can configure about your app and links to more information if necessary.

Package.appxmanifest

The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties.

Application	Visual Assets	Capabilities	Declarations	Content URIs	Packaging
<p>Display name: <input type="text" value="MyApp"/></p> <p>Entry point: <input type="text" value="MyApp.App"/></p> <p>Default language: <input type="text" value="en-US"/> More information</p> <p>Description: <input type="text" value="MyApp"/></p> <p>Supported rotations: An optional setting that indicates the app's orientation preferences.</p> <div> <input type="checkbox"/>  Landscape <input type="checkbox"/>  Portrait <input type="checkbox"/>  Landscape-flipped <input type="checkbox"/>  Portrait-flipped </div> <p>Lock screen notifications: <input type="text" value="(not set)"/></p>					

Check that you have all the images that are required for a Universal Windows app on the **Visual Assets** tab.

From the **Packaging** tab, you can enter publishing data. This is where you can choose which certificate to use to sign your app. All Universal Windows Apps must be signed with a certificate. In order to sideload an app package, you need to trust the package. The certificate must be installed on that device to trust the package.

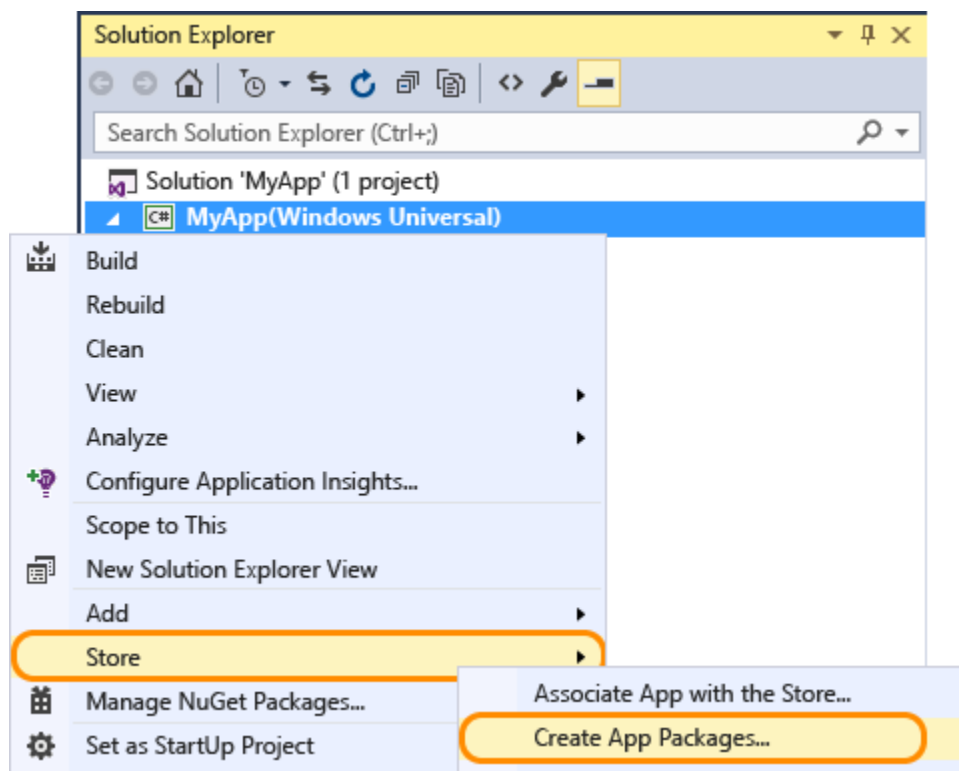
4. Save your file after you have made the necessary edits for your app.

Create an app package

To distribute an app through the Store you must create an appxupload package. You can do that by using the **Create App Packages** wizard. Follow these steps to create a package suitable for store submission with Visual Studio 2015:

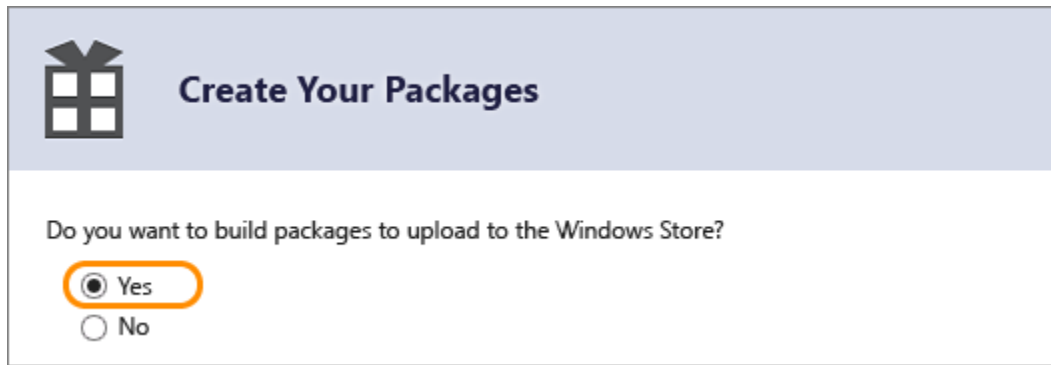
To create your app package

1. In **Solution Explorer**, open the solution for your Universal Windows app project.
2. Right-click the project and choose **Store->Create App Packages**. If this option is disabled or does not appear at all, check that the project is a Universal Windows project.



The **Create App Packages** wizard appears.

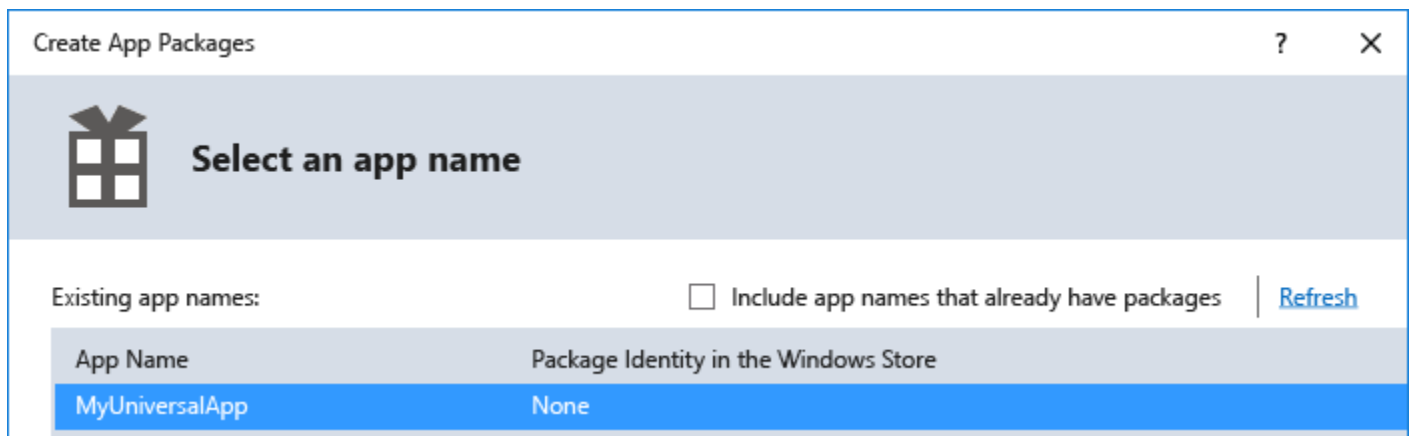
3. Select Yes in the first dialog asking if you want to build packages to upload to the Windows Store, then click Next.



The dialog box titled "Create Your Packages" features a gift icon. It asks, "Do you want to build packages to upload to the Windows Store?" with two radio button options: "Yes" (which is selected and highlighted with an orange circle) and "No".

If you choose **No** here, Visual Studio will not generate the required .appxupload package you need for store submission. If you only want to sideload your app to run it on internal devices, then you can select this option.

4. Sign in with your developer account to the Windows Dev Center. (If you don't have a developer account yet, the wizard will help you create one.)
5. Select the app name for your package, or reserve a new one if you have not already reserved one with the Windows Dev Center portal.




The "Create App Packages" dialog box, titled "Select an app name", includes a gift icon. It shows a list of "Existing app names:" with a checkbox for "Include app names that already have packages" and a "Refresh" link. Below is a table with two columns: "App Name" and "Package Identity in the Windows Store".

App Name	Package Identity in the Windows Store
MyUniversalApp	None

6. Make sure you select all three architecture configurations (x86, x64, and ARM) in the **Select and Configure Packages** dialog. That way your app can be deployed to the widest range of devices. In the **Generate app bundle** listbox, select **Always**. This makes the store submission process much simpler because you will only have one file to upload (.appxupload). The single bundle will contain all the necessary packages to deploy to devices with each processor architecture.

Create App Packages

 **Select and Configure Packages**

Output location:
 ...

Version:
 . . .
☒ Automatically increment

Generate app bundle:
 ▾
[What does an app bundle mean?](#)

Select the packages to create and the solution configuration mappings:

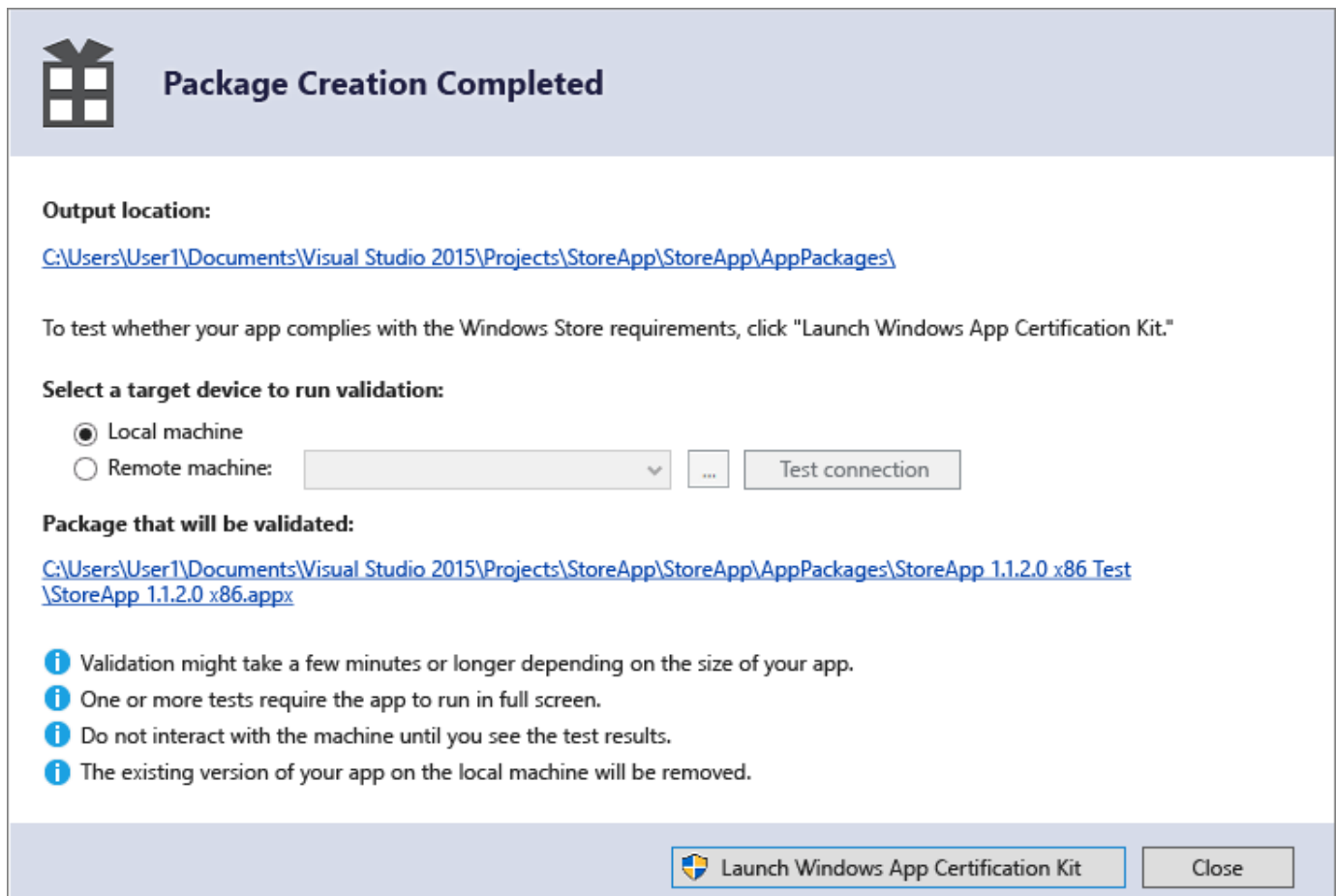
	Architecture	Solution Configuration
<input type="checkbox"/>	Neutral	None
<input checked="" type="checkbox"/>	x86	Release (x86) ▾
<input checked="" type="checkbox"/>	x64	Release (x64) ▾
<input checked="" type="checkbox"/>	ARM	Release (ARM) ▾

i The Windows Store will only accept the generated .appxupload package. Any other .appx packages are created for testing purposes only.

☒ Include full PDB symbol files, if any, to enable crash analytics for the app. [Learn More](#)

Previous Create Cancel

7. It is a good idea to include full PDB symbol files for the best [crash analytics](#) experience from the Windows Dev Center.
8. Now you can configure the details to create your package. When you're ready to publish your app, you'll upload the packages from the output location.
9. Click **Create** to generate your appxupload package.
10. Now you will see this dialog:



Validate your app before you submit it to the Store for certification on a local or remote machine. (You can only validate release builds for your app package and not debug builds.)

11. To validate locally, leave the **Local machine** option selected and click **Launch Windows App Certification Kit**.

The Windows App Certification Kit performs tests and shows you the results.

If you have a remote Windows 10 device, that you want to use for testing, you will need to install the Windows App Certification Kit manually on that device. The next section will walk you through these steps. Once you've done that, then you can select **Remote machine** and click **Launch Windows App Certification Kit** to connect to the remote device and run the validation tests.

12. After WACK has finished and your app has passed, you are ready to upload to the store. Make sure you upload the correct file. It can be found in the root folder of your solution [AppName]\AppPackages and it will end with .appxupload file extension. The name will be of the form [AppName]_[AppVersion]_x86_x64_arm_bundle.appxupload.

Validate your app package on a remote Windows 10 device

1. Enable your Windows 10 device for development using [these instructions](#).
2. [Download and install the remote tools](#) for Visual Studio. These tools are used to run the Windows App Certification Kit remotely.

3. [Download the required version of the Windows App Certification Kit](#) and then install it on your remote Windows 10 device.
4. On the **Package Creation Completed** page of the wizard, choose the **Remote Machine** option button, and then choose the ellipsis button next to the **Test Connection** button.
5. Specify a device from inside your subnet, or provide the Domain Name Server (DNS) name or IP address of a device that's outside of your subnet.
6. In the **Authentication Mode** list, choose **None** if your device doesn't require you to log onto it by using your Windows credentials.
7. Choose the **Select** button, and then choose the **Launch Windows App Certification Kit** button.

If the remote tools are running on that device, Visual Studio connects to it and then performs the validation tests.

[Sideload your app package](#)

With Universal Windows app packages, you cannot simply install an app to your device like Desktop apps for example. Typically, you download these apps from the Store and that is how they are installed on your device. But you can sideload apps to your device without submitting them to the Store. This lets you install them and test them out using the app package (.appx) that you have created. If you have an app that you don't want to sell in the Store, like a line-of-business (LOB) app, you can sideload that app so that other users in your company can use it.

To sideload your app package to a Windows 10 device, follow these steps:

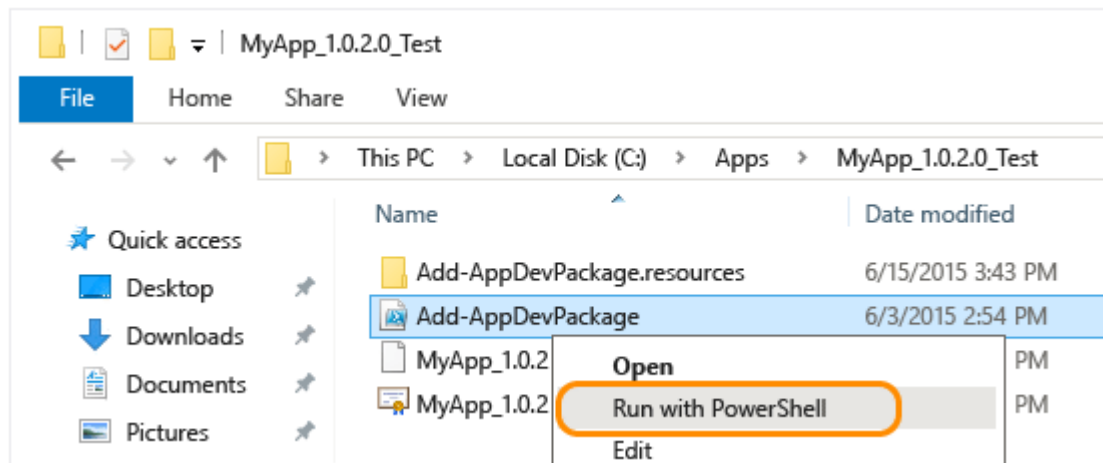
- [Enable your device](#)
- To install your app to a desktop, laptop, or tablet, follow the steps in the section below.

To install an app to a Windows 10 Mobile device, use the [WinAppDeployCmd.exe](#).

After you have sideloaded your app to test it, you can [upload your package to sell your app in the Store](#), or you can sideload your app to any Windows 10 device.

Install an app to a desktop, laptop, or tablet

1. Copy the folders for the version that you want to install to the target device.
If you've created an app bundle, then you will have a folder based on the version number and an _test folder. For example these two folders (where the version to install is 1.0.2):
 - C:\Projects\MyApp\MyApp\AppPackages\MyApp_1.0.2.0
 - C:\Projects\MyApp\MyApp\AppPackages\MyApp_1.0.2.0_TestIf you don't have an app bundle, then you can just copy the folder for the correct architecture and the corresponding test folder. For example these two folders:
 - C:\Projects\MyApp\MyApp\AppPackages\MyApp_1.0.2.0_x64
 - C:\Projects\MyApp\MyApp\AppPackages\MyApp_1.0.2.0_x64_Test
2. On the target device, open the test folder. For example: C:\Projects\MyApp\MyApp\AppPackages\MyApp_1.0.2.0_Test.
3. Right-click the **Add-AppDevPackage.ps1** file, then choose **Run with PowerShell** and follow the prompts.



When the app package has been installed, you will see this message in your PowerShell window:
Your app was successfully installed.

4. Click the Start button and then type the name of your app to launch it.

Design basics

These articles introduce you to designing a Universal Windows Platform (UWP) app, a type of Windows app built using the Windows Runtime APIs.

A Universal Windows Platform (UWP) app can run on any Windows-based device, from your phone to your tablet or PC. You can even create UWP apps that run on compact devices, such as wearables or household appliances.

When you design a UWP, you create a user interface that suits a variety of devices with different display sizes. To make that easier, we give you a set of universal controls that automatically work well on all devices. The platform does the work behind the scenes to ensure that text and visuals scale between devices and are always legible.

You can use the same code and design for all devices, and you can also tailor the user interface for specific screen sizes. For example, you can design an interface that works great for tablets and PCs and create a customized experience for mobile devices, while still reusing most of your code.

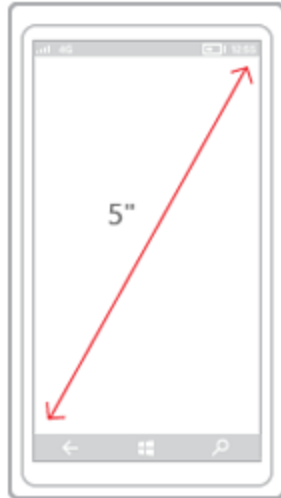
Looking for Windows 8.1 design guidance? You can get it as a PDF: [Download the Windows 8.1 guidelines](#).

Before you begin



[Intro to UWP apps for designers](#)

This article describes the features and benefits of the Universal Windows Platform from a design perspective. Find out what the platform gives you for free and the tools it offers.



[Device primer](#)

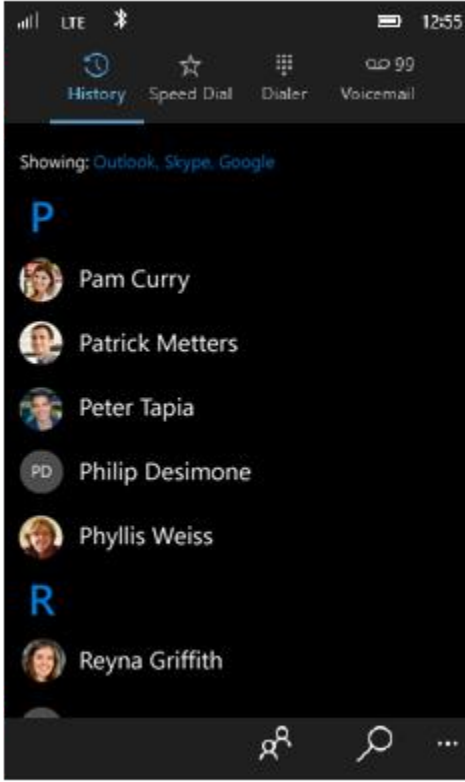
Get to know the devices that your app can run on

UI basics for Universal Windows Platform (UWP) apps

A modern user interface is a complex thing, made up of text, shapes, colors, and animations which are ultimately made up out of individual pixels of the screen of the device you're using. When you start designing a user interface, the sheer number of choices can be overwhelming.

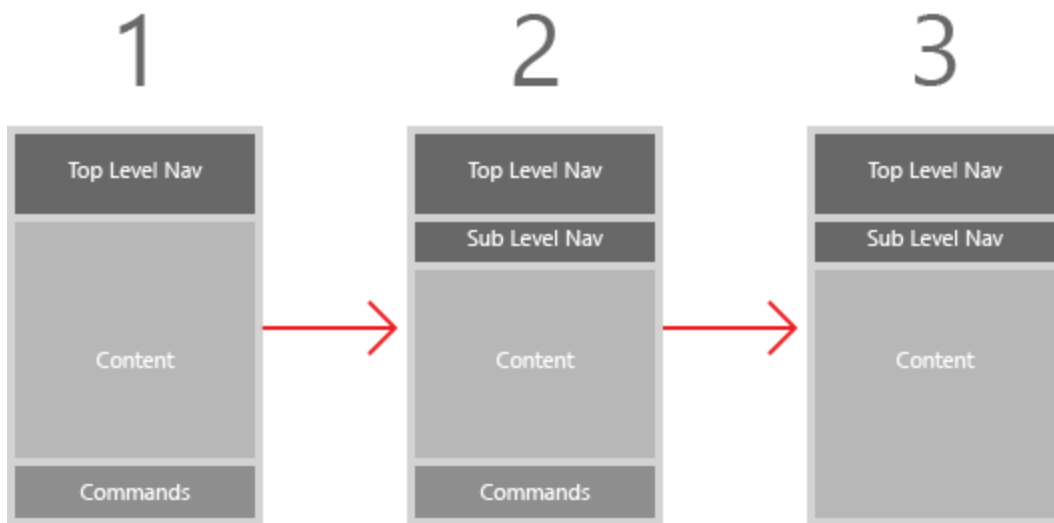
To make things simpler, let's define the anatomy of an app from a design perspective. Let's say that an app is made up of screens and pages. Each page has a user interface, made up of three types of UI elements: navigation, commanding, and content elements.

Anatomy of an app

	<p>Navigation elements</p> <p>Navigation elements help users choose the content they want to display. Examples of navigation elements include tabs and pivots, hyperlinks, and nav panes. Navigation elements are covered in detail in the Navigation design basics article.</p> <p>Command elements</p> <p>Command elements initiate actions, such as manipulating, saving, or sharing content. Examples of command elements include button and the command bar. Command elements can also include keyboard shortcuts that aren't actually visible on the screen. Command elements are covered in detail in the Command design basics article.</p> <p>Content elements</p> <p>Content elements display the app's content. For a painting app, the content might be a drawing; for a news app, the content might be a news article. Content elements are covered in detail in the Content design basics article.</p>
--	---

At a minimum, an app has a splash screen and a home page that defines the user interface. A typical app will have multiple pages and screens, and navigation, command, and content elements might change from page to page.

The following figure shows a hypothetical app structure with an assortment of pages, each of which has a different assortment of navigation, command, and content elements:



Let's take a look at some common UI patterns for combining navigation, command, and content elements.

Build UWP apps with Visual Studio

- Get started with Universal Windows apps
- Add universal controls that adapt to your Windows 10 devices
- Preview your pages on different devices
- Run your app and debug your code
- Add platform-specific code
- Handle different orientations or screen dimensions
- Create a device-specific view for a page
- Port existing apps
- Release notes for Visual Studio 2015
- Q&A
- Related topics

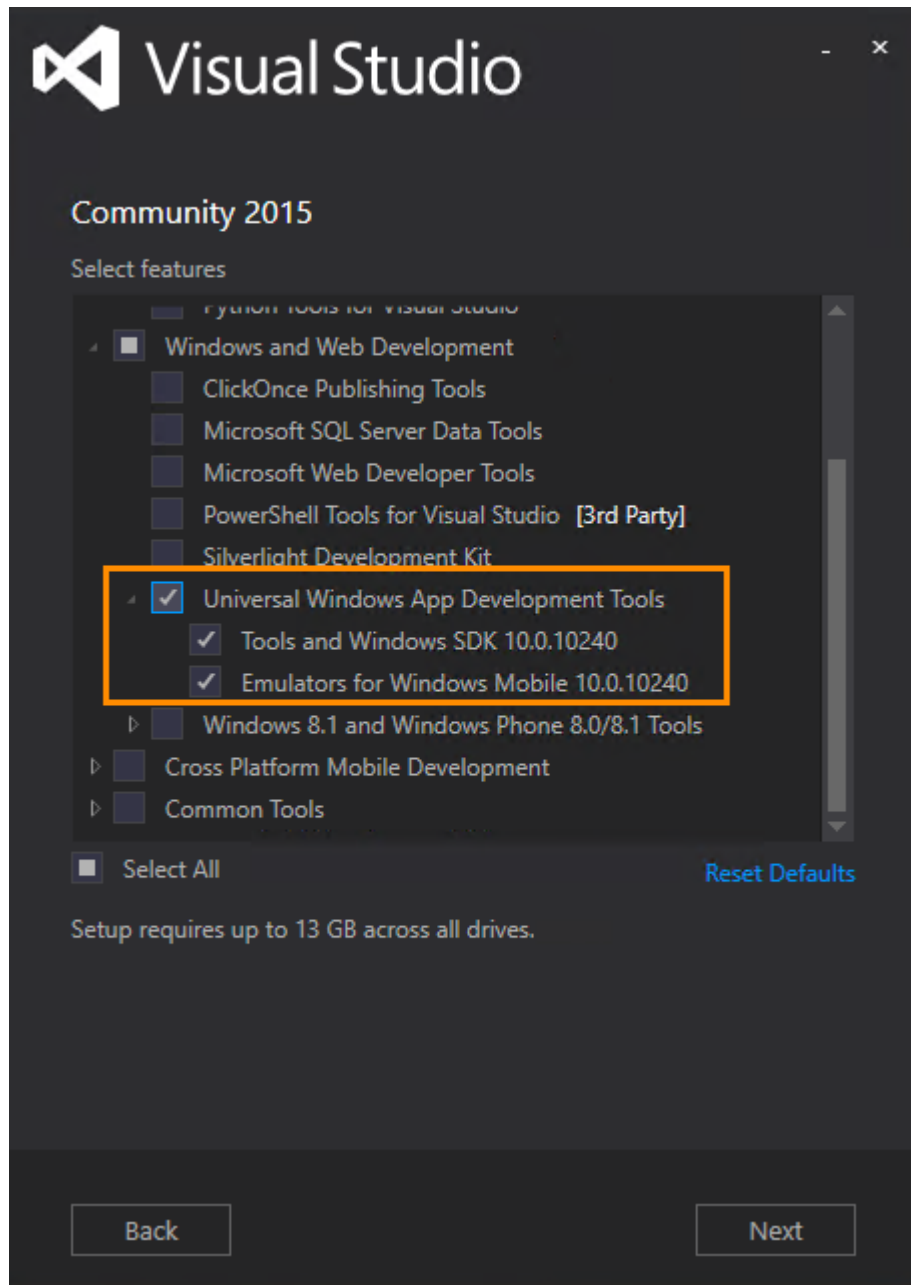
The Universal Windows app experience has improved significantly for Windows 10 from the Windows 8.1 experience. This topic explains how you create Universal Windows apps for Windows 10 devices and covers some of the differences.

With the introduction of the single, unified Windows 10 core and the Universal Windows Platform (UWP), one app package can run across all platforms. You now build one Universal Windows app that runs on all Windows 10 devices. Run your app on a Windows 10 phone, a Windows 10 desktop, or Xbox. It's the same app package! You can design your pages so they render properly no matter what device is used to view them.

If you want to get a sense of what's possible, have a look at this [UWP guide](#). Then, go ahead and build your first truly universal Windows app.

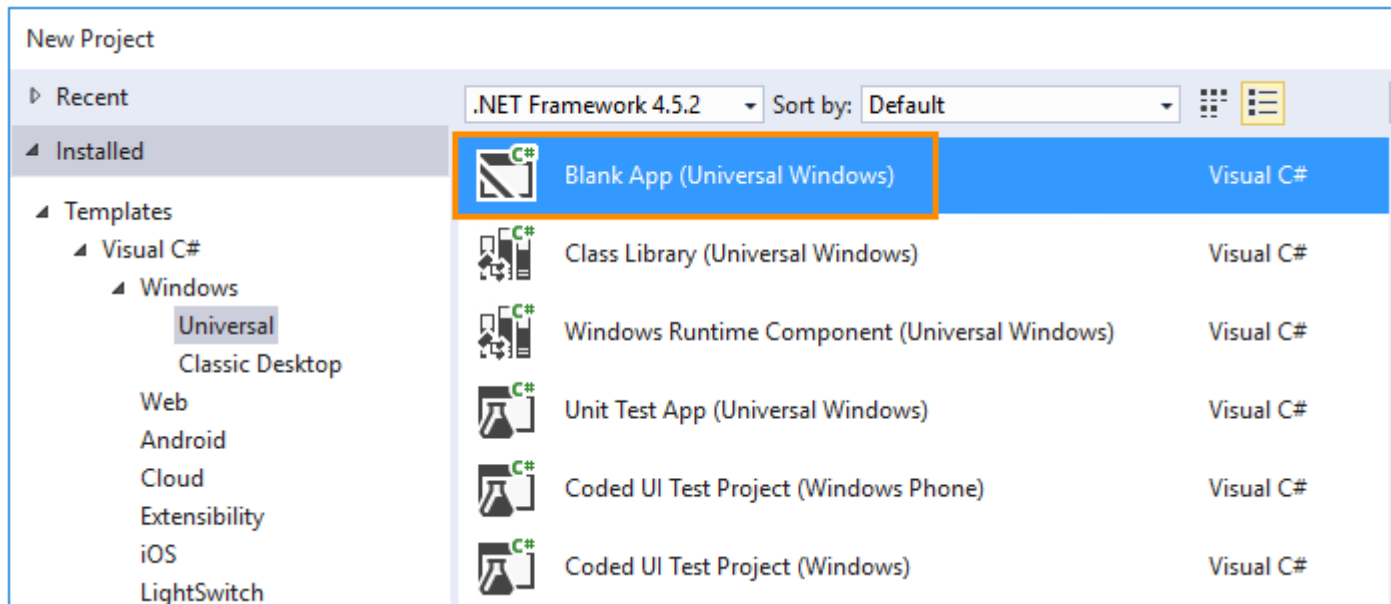
Get started with Universal Windows apps

First [download](#) and do a custom install of Microsoft Visual Studio. Make sure that the Universal Windows App Development Tools are selected from the optional features list. Without these tools, you won't be able to create your Windows 10 universal apps.



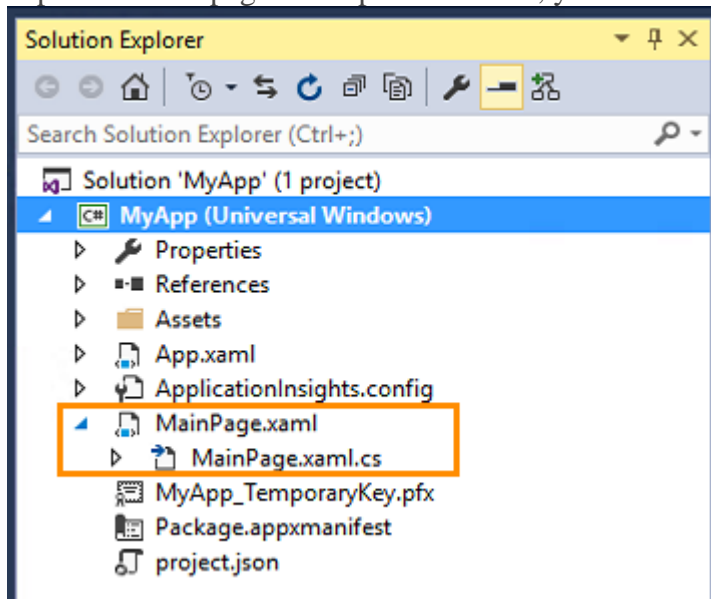
When you develop apps or test an app on a Windows 10 device, you must enable that device for development. You no longer need a developer license with Windows 10. Follow the steps [here](#) to enable your device.

Now you are ready! Choose the template based on the language that you want to use: C#, Visual Basic, C++ or JavaScript. (In this topic, we are using a C# template.) Next create a **Blank App (Universal Windows)** project.



When you've done that, notice that only one project appears in **Solution Explorer**. That's because your app is now truly adaptive and you only need one project to build it.

The project contains one page but you can add others. Each page renders on all devices that your project targets so you don't have to create multiple versions of them. If you want to optimize the experience of a page for a specific device, you most certainly can. We'll discuss that shortly.



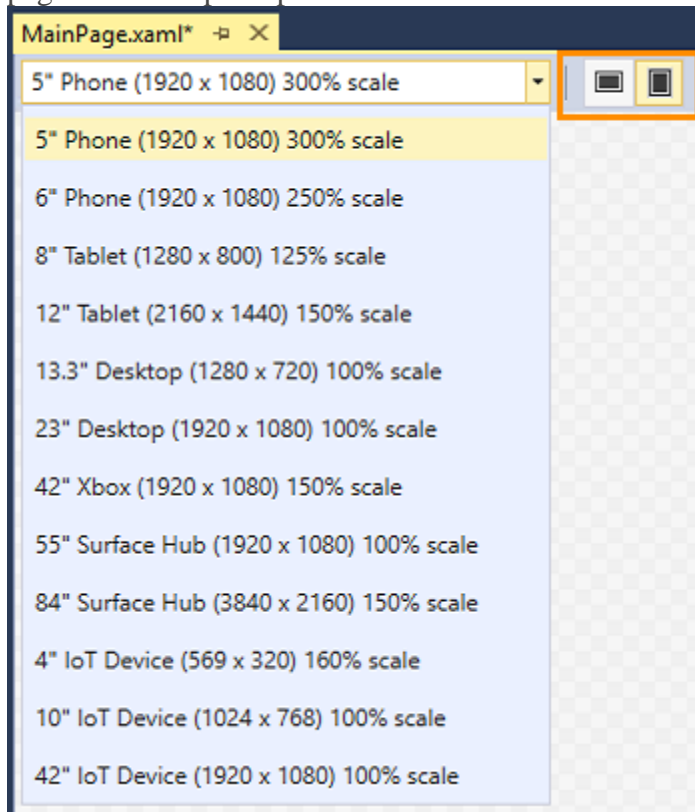
Add universal controls that adapt to your Windows 10 devices

The UWP introduces some new controls, such as the *RelativePanel* and *SplitView* controls, that adapt their appearance for different types of devices, layouts, and device orientations.

[Learn more about adaptive controls.](#)

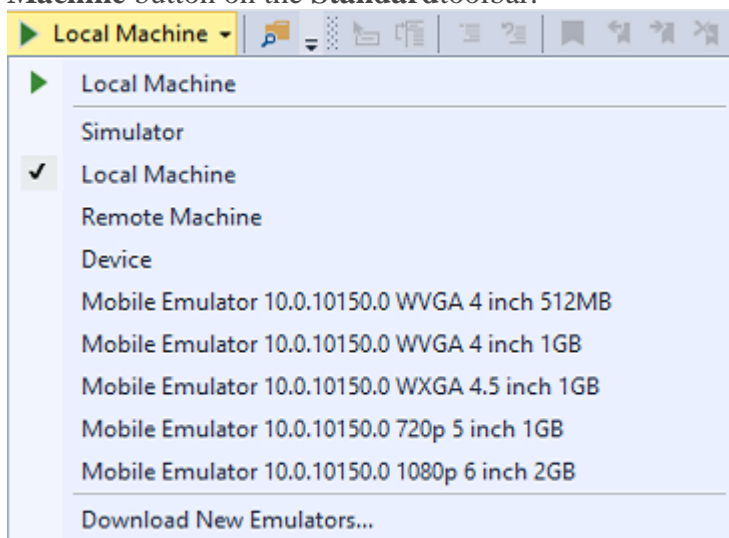
Preview your pages on different devices

You can see how a page renders on different devices without having to run your app on each device. Choose a device from a drop-down list at the top of your designer. The dimensions of the page that appears in your designer change with each selection. You can also choose to view your page in landscape or portrait mode.



Run your app and debug your code

To run your app, choose a device target from the drop-down list next to the **Local Machine** button on the **Standard** toolbar.



To see how your app appears on a tablet or desktop, run it locally, on a remote machine, or in a simulator. The simulator helps you simulate common touch and rotation events.

To see how your app appears on a different Windows device, choose an emulator or attach a device directly to your computer. To run the app with any of the emulators, you need to install Visual Studio on a physical machine. The physical machine must run Windows 8.1 (x64) Professional edition or higher, and have a processor that supports Client Hyper-V and Second Level Address Translation (SLAT). The emulators cannot run when Visual Studio is installed on a virtual machine.

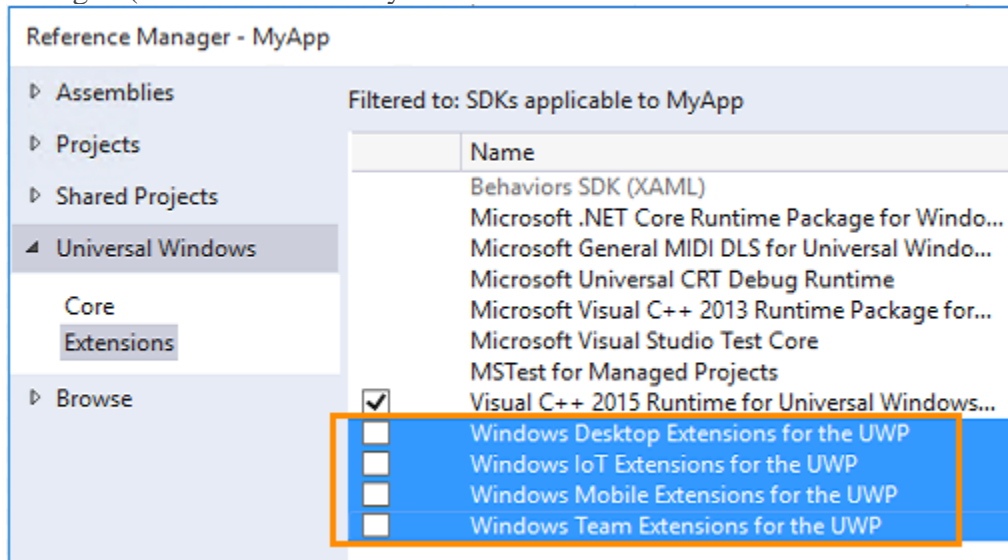
[Learn more about how to run your app in each of these device targets.](#)

[Learn more about how to debug and test your app in Visual Studio](#)

Add platform-specific code

Most of your code will run on both PCs and mobile devices. Most likely, you won't have to do anything to make that happen. That's because most APIs have been converged into a single Universal Windows SDK. You can find that SDK in your list of project references in Solution Explorer.

However, sometimes, you'll have to write code that runs only on a specific device family. To do that, open the **Reference Manager** dialog box, and then choose one of the Extension SDKs for the Universal Windows Platform (UWP) depending on the device families that your code needs to target. (Over time there may be more extension SDKs added for other device families.)



You'll have to make sure that an API is available on the running device before you use it. For example, this code assigns an event handler to the **Windows.Phone.UI.Input.HardwareButtons.CameraPressed** event. The code compiles if your project contains a reference to the **Windows Mobile Extensions for the UWP** SDK.

C#

```
Windows.Phone.UI.Input.HardwareButtons.CameraPressed
HardwareButtons_CameraPressed;
```

+=

But what happens when a user runs this app on a PC without a camera button? The app would crash because a PC does not have a back button.

To address this issue, use the **Windows.Foundation.Metadata.ApiInformation.IsTypePresent** method to determine whether a type named **Windows.Phone.UI.Input.HardwareButtons** is available on the running device. If it is, then the app is running on a mobile device, and the code will execute without crashing the app. The following code does that.

C#

// Note: Cache the value instead of querying it more than once.

bool isHardwareButtonsAPIPresent =

Windows.Foundation.Metadata.ApiInformation.IsTypePresent("Windows.Phone.UI.Input.HardwareButtons");

```
if (isHardwareButtonsAPIPresent)
{
    Windows.Phone.UI.Input.HardwareButtons.CameraPressed +=
        HardwareButtons_CameraPressed;
}
```

Use this pattern throughout your app to add device-specific functionality to your app experience. Handle different orientations or screen dimensions

Use adaptive triggers in your XAML to specify how controls should appear based on things like the orientation of the device. You can only add adaptive triggers using Blend and not Microsoft Visual Studio 2015.

[Learn how to add adaptive triggers.](#)

Create a device-specific view for a page

Sometimes you need more than adaptive controls and design states to achieve a tailored experience. No problem there. You can create device-specific views for your pages. they're not separate pages because they share the same code behind file.