

Building a Windows Phone 8 App

27

Mr. Watson, come here—I want to see you.

—Alexander Graham Bell

Objectives

In this chapter you'll:

- Register for a Windows Phone Dev Center account.
- Code a complete Windows Phone 8 app.
- Define an app's GUI with Visual Studio's GUI designer, **Document Outline** and **Properties** windows.
- Handle Windows Phone 8 user-interface events.
- Test a Windows Phone 8 app in the emulator on your desktop.
- Learn how to register a Windows Phone device for testing.
- Learn how to submit your apps to the Windows Phone Store for sale or free distribution.
- Learn about other popular app platforms to which you can port your Windows Phone 8 app to reach a broader audience.



- 27.1 Introduction
- 27.2 Downloading the Windows Phone 8 SDK
- 27.3 **Tip Calculator** App Introduction
- 27.4 Test-Driving the **Tip Calculator** App
- 27.5 **Tip Calculator** Technologies Overview
 - 27.5.1 Classes App and PhoneApplicationPage
 - 27.5.2 Arranging Controls with Grid
 - 27.5.3 Controls
 - 27.5.4 IDE Features
- 27.6 Building the App's GUI
 - 27.6.1 Grid Introduction
 - 27.6.2 Creating the TipCalculator Project
 - 27.6.3 Changing the Default GUI
 - 27.6.4 Adding the TextBlocks, Borders, a TextBox and a Slider
- 27.7 Adding Functionality to the App with C#
- 27.8 WAppManifest.xml
- 27.9 Windows Phone Dev Center
 - 27.9.1 Microsoft Account
 - 27.9.2 Windows Phone Dev Center Account
 - 27.9.3 Registering a Windows Phone Device for Development
- 27.10 Selling Your Apps in the Windows Phone Marketplace
 - 27.10.1 Free vs. Paid Apps
 - 27.10.2 Submitting Your Apps and In-App Products
 - 27.10.3 Monetizing Apps with Microsoft Advertising pubCenter
- 27.11 Other Popular Mobile App Platforms
- 27.12 Developer Documentation
- 27.13 Additional Windows Phone 8 Resources
- 27.14 Wrap-Up

Self-Review Exercises | Answers to Self-Review Exercises | Exercises

27.1 Introduction

In this chapter, you'll develop a simple **Tip Calculator** app for Windows Phone 8 (which from this point forward we'll abbreviate as WP8). This chapter assumes that you're already familiar with the Windows 8 UI and XAML concepts presented in Chapter 25. The WP8 SDK and the emulator used to test your WP8 apps both require 64-bit Windows 8 versions.

WP8—Microsoft's second-generation mobile phone operating system—was released in late 2012. WP8 is a pared down version of Windows 8, designed specifically for smartphones. These are *resource-constrained devices*—they have less memory and processor power than desktop computers, and limited battery life. WP8 has the same core operating systems services as Windows 8, including a common file system, security, networking, media and Internet Explorer 10 (IE10) web browser technology. However, WP8 has *only* the features necessary for smartphones, allowing them to run efficiently, minimizing the burden on the device's resources. Figure 27.1 lists some of the WP8 features.

WP8 developer features

- Support for multiple screen sizes.
- A tile-based user interface.
- Multitasking.
- WebBrowser control based on Internet Explorer 10.

Fig. 27.1 | Some WP8 developer features. (Part 1 of 2.)

WP8 developer features

- Map and navigation controls.
- Camera and photo APIs.
- Music and background audio APIs.
- Speech synthesis and recognition APIs.
- Bluetooth APIs.
- VoIP (Voice over IP) APIs for audio and video calls made over an Internet connection.
- Near Field Communication (NFC) APIs, which enable users to share content between two devices by touching them together (e.g., for making mobile payments, sharing contacts, etc.)
- Multilingual App Toolkit for Visual Studio 2012 for localizing your apps.
- In-app purchase for selling virtual goods (e.g., additional game levels, game scenery, ringtones, e-gifts, e-books, videos, music and more).
- Windows Phone Application Analysis for app monitoring, which allows you to determine the quality and performance of your app and improve it during the development process, rather than after making it available to users.
- Simulation Dashboard that shows you how your app will respond to network connectivity issues (e.g., a low bandwidth or weak connection), the lock screen, interruptions (e.g., notifications, phone calls) and more.
- Wallet, which allows you to offer users coupons, memberships, loyalty cards and more that they can collect and store in their Wallet on a WP8 device.
- And more.

Fig. 27.1 | Some WP8 developer features. (Part 2 of 2.)

27.2 Downloading the Windows Phone 8 SDK

The Windows Phone 8 SDK is included with the Visual Studio 2012 Professional edition or higher. If you do not have a full version of Visual Studio, you can download and install the Windows Phone 8 SDK for free from dev.windowsphone.com/downloadsdk. This will also install Visual Studio Express 2012 for Windows Phone, which includes all the tools you'll need to develop WP8 apps that you can submit to the Windows Phone Store. The WP8 SDK requires 64-bit Windows 8. The complete system requirements are located at:

bit.ly/WPSystemRequirements

Windows Phone 8 Emulator

The Windows Phone 8 SDK includes Visual Studio Express 2012 for Windows Phone and the **Windows Phone 8 Emulator**, which allows you to test your smartphone apps on your computer. If you're interested in testing your apps on WP8 devices, see Section 27.9. Figure 27.2 lists the WP8 Emulator features. You cannot test app features that use the compass, gyroscope or the vibration controller. For the latest information about the WP8 Emulator including installation, system requirements, features, running your apps in the emulator and more, see

bit.ly/WPEmulator

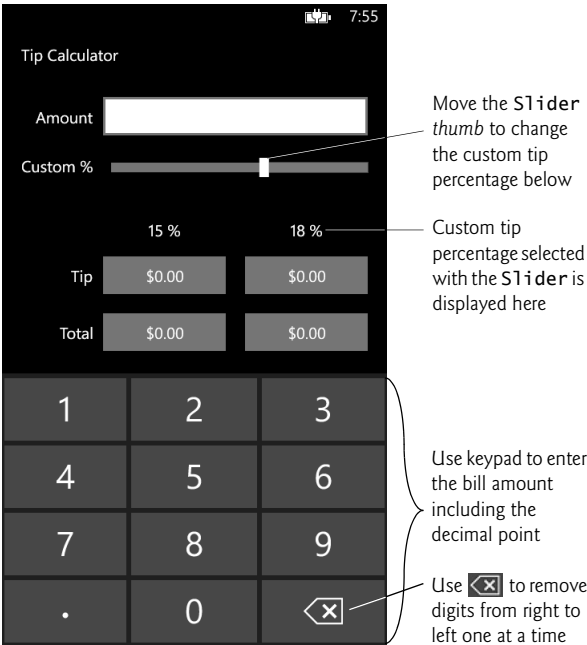
Windows Phone 8 Emulator features	
Accelerometer	App lifecycle (transitioning between active and dormant state)
Camera and video (with some limitations)	In-app purchase
GPS	Local folder (storage that's used only while the emulator is running)
Language and region settings	Memory-constrained devices
Lock screen	Multi-touch
Microphone (e.g., for speech recognition)	Networking
Multiple screen resolutions	Notifications
NFC	
Screen configuration options	
Copy and paste	

Fig. 27.2 | Windows Phone 8 Emulator features. (bit.ly/WP8EmulatorFeatures)

27.3 Tip Calculator App Introduction

The **Tip Calculator** app (Fig. 27.3(a)) calculates and displays possible tips for a restaurant bill. As you enter each digit of a bill amount by touching the *numeric keypad*, the app calculates and displays the tip amount and total bill (i.e., bill amount + tip) for a 15% tip and a custom tip percentage (18% by default). You can specify a custom tip percentage from 0% to 30% by moving the *Slider thumb*—this updates the custom percentage shown and displays the

a) Initial GUI



b) GUI after user enters the bill total 34.56 and changes the custom tip percentage to 20%

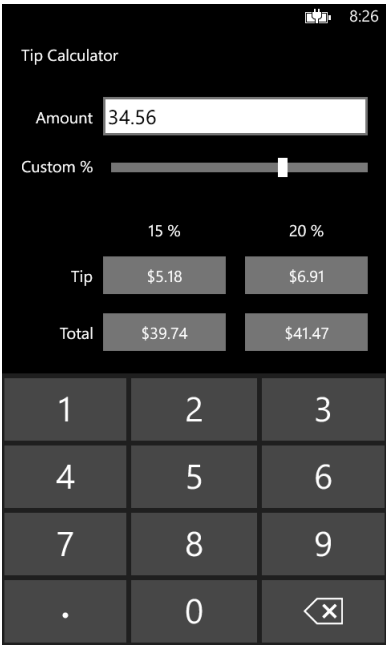



Fig. 27.3 | Entering the bill total and calculating the tip.

custom tip and total (Fig. 27.3(b)). We chose 18% as the default custom percentage because many restaurants in the U.S. add this tip percentage for parties of six people or more.


You'll begin by test-driving the app—you'll use it to calculate 15% and custom tips. Then we'll overview the technologies you'll use to create the app. You'll build the app's GUI using Microsoft Visual Studio Express 2012 for Windows Phone, which we'll refer to simply as "the IDE" throughout the rest of the chapter, and the **Document Outline** window. Finally, we'll present the complete C# code for the app and do a detailed code walkthrough.

27.4 Test-Driving the Tip Calculator App

Open and Run the App

Open the **Tip Calculator** app's project (`TipCalculator.sln`) from the `ch27` folder with the book's examples. Execute the app in the *WP8 Emulator* by selecting **DEBUG > Start Debugging** or by clicking the  button on the IDE's toolbar.

Enter a Bill Total

Using the numeric keypad, enter the bill amount 34.56 (including the decimal point). If you make a mistake, press the delete () button to erase one *rightmost* digit at a time. The `TextBlocks` under the **15%** and the custom tip percentage (**18%** by default) labels show the tip amount and the total bill for these tip percentages. All the **Tip** and **Total** `TextBlocks` update each time you enter or delete a digit.

Select a Custom Tip Percentage

Use the `Slider` to specify a *custom* tip percentage. Drag the `Slider`'s *thumb* until the custom percentage reads **20%** (Fig. 27.3(b)). As you drag the *thumb*, the tip and total for the custom tip percentage update continuously. By default, the `Slider` allows you to select values from 0.0 to 10.0—we specified a maximum value of 0.3 (representing a 30% tip) for this app.

27.5 Tip Calculator Technologies Overview

This section introduces the IDE features and WP8 technologies you'll use to build the **Tip Calculator** app. We assume that you're familiar with the Windows 8 UI concepts presented in Chapter 25.

27.5.1 Classes App and PhoneApplicationPage

Unlike many C# apps, WP8 apps *do not have a Main method*. Instead, they have a derived class of `Application` (namespace `System.Windows.Application`) that the OS uses to launch an app. This class, named `App`, is defined by the files `App.xaml` and `App.xaml.cs`. As in a Windows Store app, the `App` class defines app-level event handlers (for events like the `Launching`, `Activated`, `Deactivated` and `Closing`), app-wide resources and more. The app-level event handlers are defined in class `App`'s code-behind file (`App.xaml.cs`)—you can add your own code to these event handlers to respond to any of these events.

A WP8 app consists of one or more *pages* that are derived classes of `PhoneApplicationPage` (namespace `Microsoft.Phone.Controls`). As in Windows 8 UI, each page's XAML specifies the *controls* that enable the user to interact with the page, and each page's code-behind file defines the page's functionality. You'll build the **Tip Calculator**'s GUI *without* manipulating the XAML directly.

27.5.2 Arranging Controls with Grid

Recall that layouts arrange controls in an GUI. As you did in Chapter 25, you'll use a `Grid` (namespace `System.Windows.Controls`) to arrange controls into rows and columns. You'll use the `Grid`'s `RowDefinitions` and `ColumnDefinitions` properties to define the rows and columns. You'll also specify two controls that span multiple columns.

27.5.3 Controls

You'll create `TextBlocks`, `Borders`, a `TextBox` and a `Slider` in this app:

- A **TextBox**—often called a *text field* in other GUI technologies—can display text, but is normally used to receive text input from the user. You'll specify a `TextBox` that allows only *numeric* input representing the bill amount.
- A **Border** enables you to place a border around a control or group of controls. You'll use `Borders` in this app to place a filled rectangle of color behind the `TextBlocks` that display the calculated tips and totals.
- A **TextBlock** displays text.
- A **Slider** represents a `double` in the range 0.0–10.0 by default and allows the user to select a number in that range by moving the `Slider`'s *thumb*. You'll customize the `Slider` so the user can choose a custom tip percentage *only* from the more limited range 0.0 to 0.30 (for tips of 0 to 30%).

27.5.4 IDE Features

You'll use the IDE's GUI designer, **Document Outline** window and **Properties** window to create, organize and customize control properties. The **Document Outline** window shows the *nested structure* of a page's layouts and controls and makes it easier to select specific controls to customize them with the **Properties** window.

27.6 Building the App's GUI

In this section, we'll walk through the precise steps for building the **Tip Calculator**'s GUI. The GUI will not look exactly like Fig. 27.3 until you've completed all the steps.

27.6.1 Grid Introduction

This app uses a `Grid` (Fig. 27.4) to arrange controls into five *rows* and two *columns*. Rows and columns are indexed from 0, like arrays. Each cell in a `Grid` can be *empty* or can hold one or more *controls*, including layouts that *contain* other controls. Controls can span *multiple* rows or columns, as shown in row 0 and row 1. As you'll see, you can specify a `Grid`'s number of rows and columns.

Row Heights and Column Widths in a Grid

Each row's *height* and each column's *width* can be specified as an *explicit size*, a *relative size* (using `*`) or `Auto`. `Auto` makes the row or column only as big as it needs to be to fit its contents. The setting `*` specifies the size of a row or column with respect to the `Grid`'s other rows and columns. For example, a column with a `Height` of `2*` would be twice the size of

	column 0	column 1	column 2	
row 0	Amount	<input type="text"/>	<input type="text"/>	In each of these two rows, a control spans two columns
row 1	Custom %	<input type="range"/>		
row 2		15 %	18 %	
row 3	Tip	\$0.00	\$0.00	
row 4	Total	\$0.00	\$0.00	

Fig. 27.4 | Tip Calculator GUI's Grid labeled by its rows and columns.

a column that is 1* (or just *). A Grid first allocates its space to the rows and columns that *explicitly* define their sizes or that are configured to allow the layout to *automatically* determine their sizes. The remaining space is divided among the other rows and columns.

In Fig. 27.4, the first column's width is set to Auto, so its width is determined by the TextBlock containing **Custom %**—the widest control in that column. We did not set the width of the second and third columns, so both columns occupy 50% of the remaining available horizontal space. All of this Grid's rows have their heights set to Auto. You'll see how to configure rows and columns in Section 27.6.3—for example, you can specify the exact row and column in which a control is to be placed.

x:Name Property Values for This App's Controls

Figure 27.5 shows the controls' **x:Name** property values—in the **Properties** window, this is represented as the **Name** property. Recall from Chapter 25 that **x:Name** values are used as the control's variable names in your C# code. For clarity, our naming convention is to use the control's class name in the **x:Name** property. In the first and second rows, the `amountTextBox` and `customTipPercentSlider` each span two columns.

amountTextBlock	Amount	<input type="text"/>	amountTextBox
customPercentTextBlock	Custom %	<input type="range"/>	customTipPercentSlider
percent15TextBlock	15 %	18 %	percentCustomTextBlock
tipTextBlock	Tip	\$0.00	tipCustomTextBlock
tip15TextBlock			
totalTextBlock	Total	\$0.00	totalCustomTextBlock
total15TextBlock			

Fig. 27.5 | Tip Calculator GUI's components labeled with their **x:Name** property values.

27.6.2 Creating the TipCalculator Project

In this section, you'll create a new project for a WP8 app using the **Windows Phone App** template provided by the IDE. This template represents a simple *single-screen app*. There are 10 other templates for various common scenarios found in many types of apps—clicking a given template name shows a preview screen and a brief description of that template's purpose. Each template saves you time by providing capabilities for common app scenarios.

Perform the following steps to create the project:

1. Open Visual Studio Express 2012 for Windows Phone.
2. Select **File > New Project...** to display the **New Project** dialog.
3. In the dialog under **Templates > Visual C#**, select **Windows Phone** to display the list of Windows Phone project templates.
4. Select the **Windows Phone App** template.
5. In the **Name:** field, enter **TipCalculator**.
6. In the **Location:** field, specify where you'd like to save the app's project—you can change this by clicking the **Browse...** button and selecting a folder in the **Project Location** dialog.
7. Leave the default value in the **Solution name:** field—by default, the solution's name is the same as the project's name.
8. Click **OK**.
9. In the **New Windows Phone Application** dialog, ensure that **Windows Phone OS 8.0** is selected, then click **OK**.

The IDE creates and configures the **TipCalculator** project, then displays the app's **MainPage.xaml** file so that you can begin designing your app's GUI. In Fig. 27.6, we collapsed the **Solution Explorer** window and set the GUI designer's **Zoom** to 50% so that the screen capture would fit in the book. The initial GUI provided by the **Windows Phone App** template contains two **TextBlocks**—you'll modify the first and remove the second in the next section.

Device Window

At the left side of the IDE, the **Device** window (**Design > Device Window**) allows you to specify various settings for the GUI designer, including:

- **Orientation**—Allows you to choose between portrait and landscape for the design area.
- **Display**—Allows you to choose the screen resolution for which you're designing the GUI.
- **Theme**—Allows you to choose between the default **Dark** theme with white text and a **Light** theme with dark text.
- **Accent**—Allows you to set the accent color for your app in the GUI designer. Windows phone allows users to choose from one of several accent colors and many controls use the currently selected color so that they have the same look and feel as other apps on the device. The default accent color is **Red** and you can choose other colors to see how your app will appear with each accent color.

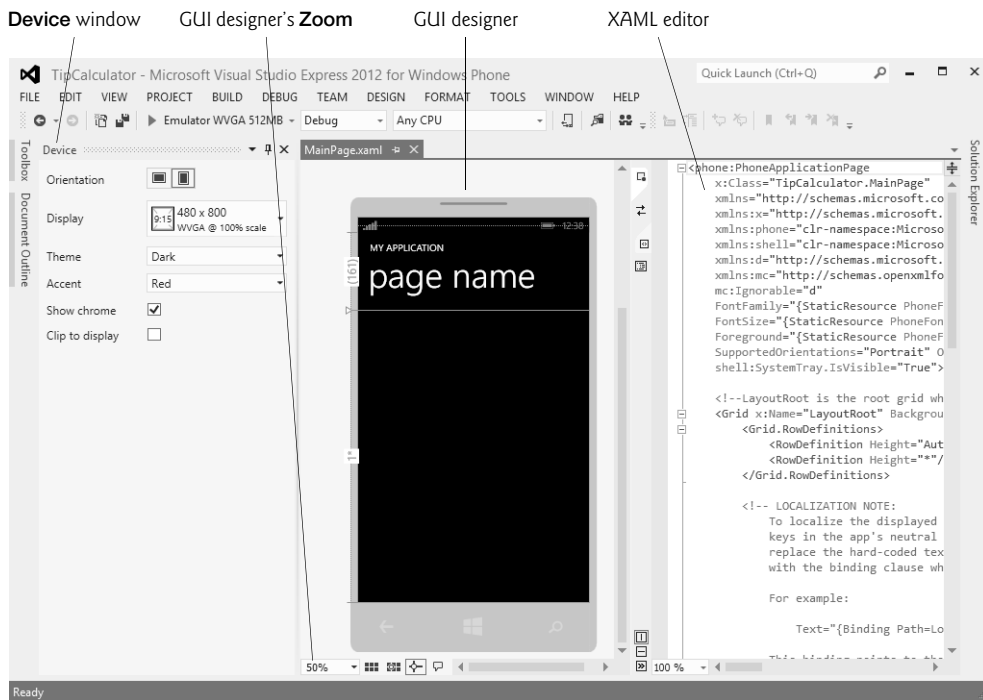


Fig. 27.6 | TipCalculator project in the IDE.

27.6.3 Changing the Default GUI

The default MainPage layout consists of a Grid with two rows—the first row contains a vertically oriented StackPanel named TitlePanel and the second row contains a Grid named ContentPanel in which you define the page's GUI. In the StackPanel are two TextBlocks—one displays **MY APPLICATION** and the other **page name**. In apps with multiple screens, these help provide context to users so they know which app they're using and which page of the app they're interacting with.

Changing the App Name

Change **MY APPLICATION** to **Tip Calculator** by double clicking the text in the designer and typing the new value directly in the designer. You can also select the control in the designer, then change its **Text** property in the **Common** section of the **Properties** window.

Deleting the Page Name

This app has only one screen, so we don't need the page name to provide additional context to the user. For that reason, select the TextBlock containing **page name** and delete it.

Specifying Five Rows and Three Columns

Recall that the GUI in Fig. 27.4 consists of five rows and three columns. You'll now use the **Document Outline** window to select the Grid, then use the **Properties** window to define

the rows and columns by setting the Grid's **RowDefinitions** and **ColumnDefinitions** properties. Perform the following tasks:

1. Open the **Document Outline** window by selecting **VIEW > Other Windows > Document Outline**.
2. In the **Document Outline** window, expand the **LayoutRoot** node and select **ContentPanel1**—this is the predefined Grid control (provided when you created the project) in which you'll define the rest of the **Tip Calculator**'s GUI.
3. In the **Properties** window, expand the **Layout** node, then click the **Show advanced properties** (▼) button to display all of the Grid's layout-related properties.
4. Click the ellipsis button to the right of the **RowDefinitions** property to display the **RowDefinition Collection Editor** dialog Fig. 27.7.

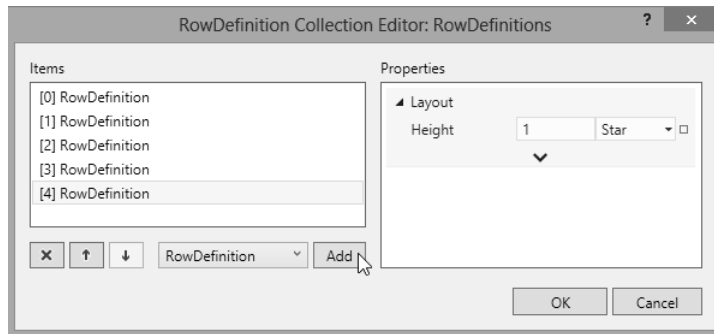


Fig. 27.7 | RowDefinition Collection Editor.

5. To create five rows, click the **Add** button five times to create five **RowDefinitions**—each allows you to specify the **Height** of a Grid row. You can also specify minimum and maximum row heights, though we do not use that in this app.
6. By default each row's **Height** property is set to 1 Star as shown for the last RowDefinition object in Fig. 27.7. One at a time, select each **RowDefinition** and set its height to **Auto**.
7. Click **OK** to dismiss the dialog.

Repeat the preceding tasks to create three **ColumnDefinitions** for the Grid's **ColumnDefinitions** property by using the **ColumnDefinition Collection Editor** dialog that's displayed when you click the ellipsis button to the right of the **ColumnDefinitions** property. For the first **ColumnDefinition**, set its **Width** to **Auto** so that the column's **Width** will be based on the widest control in the column. Leave the default **Width** for the other two **ColumnDefinitions**, so that they'll split the remaining horizontal space evenly between them.

27.6.4 Adding the TextBlocks, Borders, a TextBox and a Slider




You'll now build the rest of the GUI in Fig. 27.4. As you add each control to the GUI, immediately use the **Properties** window to set the control's **Name** property using the names shown in Fig. 27.5.

You'll use the **Document Outline** window to add controls to the Grid's rows and columns. The steps refer to the row and column numbers in Fig. 27.4. When working with layouts, it's sometimes difficult to see the layout's *nested structure* and to place controls in the correct locations by dragging them onto the GUI designer. The **Document Outline** window makes these tasks easier because it clearly shows the GUI's nested structure. As you'll soon see, you can reorder controls by dragging them in the **Document Outline** window.

The steps we provide here help you configure the GUI to appear exactly as shown in Fig. 27.4. When you build your own apps, you should experiment with the many control properties that are available to you.

Step 1: Configuring Row 0

Row 0 consists of the `amountTextBlock` in column 0 and the `amountTextBox` in columns 1 and 2—that is, it *spans* two columns. Each time you drop a control onto the GUI designer, the control is placed in the exact location you specify. You'll use the **Properties** window to specify values that place each control into the proper row and column. Perform the following tasks to create row 0's controls:

1. If the **Toolbox** window is not displayed, select **VIEW > Toolbox**.
2. Drag a `TextBlock` (`amountTextBlock`) onto the GUI designer.
3. In the **Properties** window's **Common** section, set **Text** to **Amount**.
4. In the **Properties** window's **Layout** section, set **Row** and **Column** to 0. Set the left **Margin** to 12 and the other **Margin** values to 0. Set the **HorizontalAlignment** to **Right** () and the **VerticalAlignment** to **Center** (). All of the `TextBlocks` in column 0 will have their left **Margins** set to 12 to ensure that whichever one is the *widest* will align with the app name (**Tip Calculator**) that's displayed at the top of the GUI.
5. Next, drag a `TextBox` (`amountTextBox`) onto the GUI designer.
6. In the **Properties** window's **Common** section, delete the default value of **Text** and set **InputScope** to **Number**—this restricts the `TextBox` to *numeric input* and ensures that a *numeric keypad* will be displayed when this control has the focus.
7. In the **Layout** section, set **Row** to 0, **Column** to 1 and **ColumnSpan** to 2. Set all **Margin** values to 0. Set the **HorizontalAlignment** to **Stretch** ()

Step 2: Configuring Row 1

Next, you'll add a `TextBlock` and `Slider` to the Grid. To do so:

1. Drag a `TextBlock` (`customPercentTextBlock`) onto the GUI designer, then set its **Text** (**Custom %**), **Row** (1), **Column** (0) and **HorizontalAlignment** (**Right**). For the **Margin**, set the **Top** and **Left** to 12 and the **Bottom** and **Right** to 0.
2. Drag a `Slider` (`customTipPercentSlider`) onto the GUI designer, then set its **Row** (1), **Column** (1), **RowSpan** (1), **ColumnSpan** (2), **HorizontalAlignment** (**Stretch**) and **Margin** (0 for all four sides).

Step 3: Configuring Row 2

Next, you'll add two `TextBlocks` to the Grid. To do so:

1. Drag a `TextBlock` (`percent15TextBlock`) onto the GUI designer, then set its **Text** (**15 %**), **Row** (2), **Column** (1), **HorizontalAlignment** (**Center**), **Margin** (0 for all

four sides) and **Padding** (8 for all four sides)—**Padding** adds space around the content *inside* a control's borders, whereas **Margin** adds space *outside* a control's borders. Expand the **Properties** window's **Text** section, select the **Paragraph** (¶) tab and change the **TextAlignment** (≡) to Center.

2. Repeat the preceding instructions for the percentCustomTextBlock but set its **Text** to 18 % and its **Column** to 2.

Step 4: Configuring Row 3

Next, you'll add three TextBlocks and two Borders. TextBlocks are transparent, so we use Borders to add color behind the tip15TextBlock and tipCustomTextBlock. Perform the following tasks:

1. Drag a TextBlock (tipTextBlock) onto the GUI designer, then set its **Text** (Tip), **Row** (3), **Column** (0), **HorizontalAlignment** (Right) and **VerticalAlignment** (Center). For the **Margin**, set **Left** (12) and other **Margin** values to 0.
2. Drag a Border onto the GUI designer, then set its **Row** (3), **Column** (1), **RowSpan** (1), **ColumnSpan** (1), **Margins** (12 on all sides), **HorizontalAlignment** (Stretch) and **VerticalAlignment** (Stretch). In the **Properties** window's **Brush** section, select the Border's **Background** property, then click the **Brush** resources (≡) tab to view a list of predefined *color resources* from the device's *theme*, which defines the look-and-feel of the GUI. Select PhoneAccentBrush as the Border's background color. The actual color is determined at *runtime* based on the theme setting on each device. The default color in the Windows Phone Emulator is red.
3. Drag a TextBlock (tip15TextBlock) onto the GUI designer. Initially, in the **Document Outline** window, tip15TextBlock appears below the Border you just created. We'd like the tip15TextBlock to be *nested* in the Border, so that its background color is displayed *behind* tip15TextBlock. To do so, in the **Document Outline** window, drag tip15TextBlock onto the [Border] node (Fig. 27.8). The tip15TextBlock is now nested in the [Border] node.

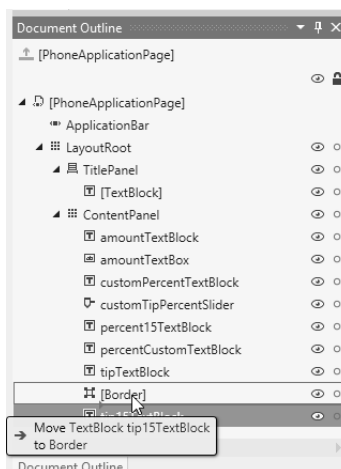


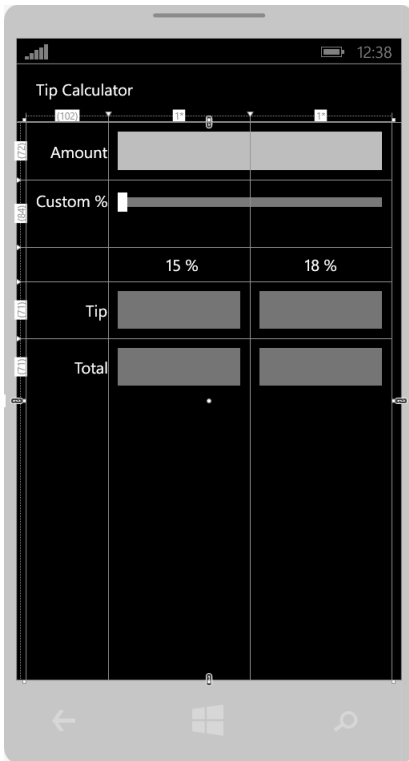
Fig. 27.8 | Dragging tip15TextBlock onto the [Border] node in the **Document Outline**.

4. With `tip15TextBlock` selected, set its **Margins** (10 for all four sides), **HorizontalAlignment** (Center), **VerticalAlignment** (Center) and **TextAlignment** (Center).
5. Repeat items 2–4 above for another **Border** and the `tipCustomTextBlock`, placing the new **Border** in row 3 and column 2.

Step 5: Configuring Row 4

Repeat Step 4 to create and configure the last row's controls, but specify row 4 for the `totalTextBlock` and the two **Borders**, and set `totalTextBlock`'s **Text** to `Total`. The GUI and **Document Outline** window should now appear as shown in Fig. 27.9.

a) Final GUI design



b) **Document Outline** window showing all the controls

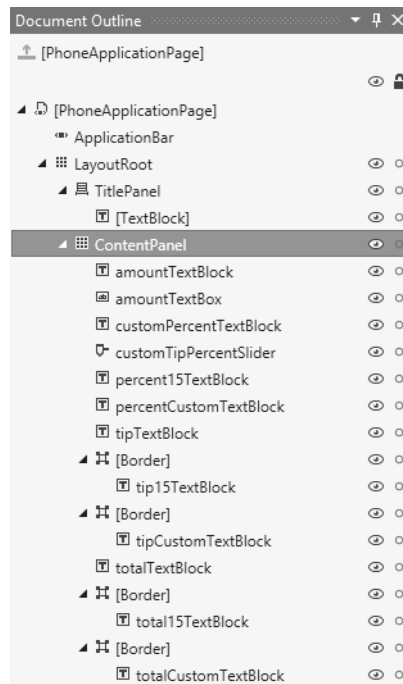


Fig. 27.9 | The GUI and the **Document Outline** window after adding all the controls.

27.7 Adding Functionality to the App with C#

Class `MainPage`'s *code-behind file* (Figs. 27.10–27.18) implements the **Tip Calculator** app's functionality. It calculates the 15% and custom percentage tips and total bill amounts, and displays them in *locale-specific* currency format.

MainPage Code-Behind File's using Statements

Figure 27.10 shows the using statements in MainPage.xaml.cs. These *namespaces* and several others were added to the code-behind file when you created the TipCalculator project. We deleted the using statements that we did not use in this app.

```

1 // MainPage.xaml.cs
2 // Calculates bills using 15% and custom percentage tips
3 using System; // for classes String and Convert
4 using System.Windows; // for event handler argument classes
5 using System.Windows.Controls; // for event handler argument classes
6 using Microsoft.Phone.Controls; // for base class PhoneApplicationPage
7

```

Fig. 27.10 | MainPage code-behind file's using statements.

MainPage Derived Class of PhoneApplicationPage

Class MainPage (Figs. 27.11–27.18) derives from class PhoneApplicationPage (Fig. 27.11) the base class for each page of your app. When you created this app's project, the IDE created this derived class for you.

```

8 namespace TipCalculator
9 {
10     public partial class MainPage : PhoneApplicationPage
11     {

```

Fig. 27.11 | Class MainPage derives from PhoneApplicationPage.

Instance Variables and the MainPage Constructor

Lines 12–13 of Fig. 27.12 declare class MainPage's variables. The bill amount entered in amountTextBox will be converted to a decimal (for use in monetary calculations) and stored in billAmount (line 12). The custom tip percentage (a double value limited to the range 0.0–0.3) that the user sets by moving the Slider thumb will be converted to a decimal (Fig. 27.17) for use in tip calculations then stored in customTipPercent (line 13).

```

12     private decimal billAmount = 0.0M; // amount entered by the user
13     private decimal customTipPercent = 0.18M; // custom tip percentage
14
15     // Constructor
16     public MainPage()
17     {
18         InitializeComponent();
19     } // end constructor
20


```

Fig. 27.12 | MainPage class's instance variables and constructor.

PhoneApplicationPage_Loaded Event Handler

The PhoneApplicationPage_Loaded event handler (Fig. 27.13) is called by the system after a PhoneApplicationPage's constructor has built the app's GUI. You use this method

to perform additional tasks that require the GUI to exist—we use this event handler to display values in the **Tip** and **Total** TextBlocks, and to customize the **Slider**’s settings, and to give the **TextBox** the focus so that the keyboard is displayed when the app appears on the screen. To create this event handler:

1. Select the [PhoneApplicationPage] node in the **Document Outline** window.
2. In the **Properties** window, click the **Event handlers** icon () to display the list of the page’s events.
3. Double click the textbox to the right of the Loaded event.
4. Insert the code in lines 25–30.

```

21      // executes when app has loaded
22      private void PhoneApplicationPage_Loaded(object sender,
23          RoutedEventArgs e)
24      {
25          // update GUI based on billAmount and customTipPercent
26          update15PercentTip(); // update the 15% tip TextBlocks
27          updateCustomTip(); // update the custom tip TextBlocks
28          customTipPercentSlider.Value = 0.18; // initial value
29          customTipPercentSlider.Maximum = 0.30; // maximum value
30          amountTextBox.Focus(); // give amountTextBox the focus
31      } // end method PhoneApplicationPage_Loaded
32

```

Fig. 27.13 | MainPage class’s PhoneApplicationPage_Loaded event handler.

Displaying Initial Values in the TextBlocks

Lines 26–27 call methods `update15PercentTip` (Fig. 27.14) and `updateCustomTip` (Fig. 27.15) to display initial values in the tip and total TextBlocks.

Customizing the Slider’s Settings

Lines 28–29 set the `customTipPercentSlider`’s **Value** property to 0.18 (that is, 18%) and its **Maximum** property to 0.30 (a maximum 30% tip).

Giving amountTextBox the Focus

Line 30 calls the `amountTextBox`’s **Focus** method to make `amountTextBox` the *active* control on the screen. When a **TextBox** *receives the focus*, the system displays an appropriate *soft keyboard* on the screen. In Section 27.6, you set the **TextBox**’s **InputScope** property to **Number**, so a keyboard that allows only numeric input is displayed in this app. Giving `amountTextBox` the *focus* as soon as the app is loaded ensures that the *numeric keyboard* is displayed so that the user can begin interacting with the app immediately.

Method update15PercentTip of Class MainPage

Method `update15PercentTip` (Fig. 27.14) is called from the `amountTextBox`’s **TextChanged** event handler to update the 15% tip and total TextBlocks each time the user *changes* the bill amount. The method uses the `billAmount` value to calculate the tip amount (line 37) and the total of the bill amount and tip (line 38). Lines 41–43 display the amounts in currency format.

```

33      // updates 15% tip TextBlocks
34      private void update15PercentTip()
35      {
36          // calculate 15% tip and total
37          decimal fifteenPercentTip = billAmount * 0.15M;
38          decimal fifteenPercentTotal = billAmount + fifteenPercentTip;
39
40          // display 15% tip and total formatted as currency
41          tip15TextBlock.Text = String.Format("{0:C}", fifteenPercentTip);
42          total15TextBlock.Text =
43              String.Format("{0:C}", fifteenPercentTotal);
44      } // end method update15PercentTip
45

```

Fig. 27.14 | MainPage method update15PercentTip calculates and displays the 15% tip and total.

Method updateCustomTip of Class MainPage

Method updateCustomTip (Fig. 27.15) is called from the customTipPercentSlider's ValueChanged event handler to update the tipCustomTextBlock and totalCustomTextBlock based on the customTipPercent. Lines 50–51 set percentCustomTextBlock's text to the customTipPercent value formatted as a percentage. Lines 54–55 calculate the customTip and customTotal. Then, lines 58–59 display the amounts in currency format.

```

46      // updates the custom tip and total TextBlocks
47      private void updateCustomTip()
48      {
49          // show customTipPercent in percentCustomTextBlock formatted as %
50          percentCustomTextBlock.Text =
51              String.Format("{0:P0}", customTipPercent);
52
53          // calculate the custom tip and total
54          decimal customTip = billAmount * customTipPercent;
55          decimal customTotal = billAmount + customTip;
56
57          // display custom tip and total formatted as currency
58          tipCustomTextBlock.Text = String.Format("{0:C}", customTip);
59          totalCustomTextBlock.Text = String.Format("{0:C}", customTotal);
60      } // end method updateCustomTip
61

```

Fig. 27.15 | MainPage method updateCustomTip calculates and displays customTip and customTotal.

amountTextBox_TextChanged Event Handler

The amountTextBox_TextChanged event handler (Fig. 27.16) is called whenever the user *modifies* the text in the amountTextBox. To create this event handler, double click the amountTextBox in the GUI designer then add the code in lines 66–78. Line 69 attempts to convert the user input in amountTextBox to a decimal and assign it to billAmount. An exception will occur if the TextBox is *empty*, in which case billAmount is set to 0.0M (line

73). Lines 77–78 call methods `update15PercentTip` and `updateCustomTip` to recalculate and display the tips and totals.

```

62      // updates 15% tip and total when user enters text in amountTextBox
63      private void amountTextBox_TextChanged(object sender,
64          TextChangedEventArgs e)
65      {
66          // convert amountTextBox's text to a decimal
67          try
68          {
69              billAmount = Convert.ToDecimal(amountTextBox.Text);
70          } // end try
71          catch (FormatException)
72          {
73              billAmount = 0.0M; // default if an exception occurs
74          } // end catch
75
76          // display currency formatted bill amount
77          update15PercentTip(); // update the 15% tip TextBlocks
78          updateCustomTip(); // update the custom tip TextBlocks
79      } // end method amountTextBox_TextChanged
80

```

Fig. 27.16 | MainPage class's `amountTextBox_TextChanged` event handler.

customTipPercentSlider_ValueChanged Event Handler

The `customTipPercentSlider_ValueChanged` event handler (Fig. 27.17) is called when the user moves `customTipPercentSlider`'s *thumb*. To create this event handler, double click the `customTipPercentSlider` in the GUI designer, then add the code in lines 85–88. Lines 86–87 get the `Slider`'s current `Value`, convert it to a decimal and *round* it to two digits after the decimal point. Rounding ensures that `customTipPercent` always represents a whole number percentage. In this app, the `Slider`'s `Value` can be *any* value in the range 0.0–0.3, including one like 0.21749567, which would represent 21.749567%. Line 88 calls method `updateCustomTip` to re-calculate and display the custom tip and total.

```

81      // updates custom tip and total when user changes slider value
82      private void customTipPercentSlider_ValueChanged(object sender,
83          RoutedPropertyChangedEventArgs<double> e)
84      {
85          // sets customTipPercent to position of the Slider's thumb
86          customTipPercent = Math.Round(
87              Convert.ToDecimal(customTipPercentSlider.Value), 2);
88          updateCustomTip(); // update the custom tip TextBlocks
89      } // end method customTipPercentSlider_ValueChanged
90

```

Fig. 27.17 | MainPage class's `customTipPercentSlider_ValueChanged` event handler.

amountTextBox_LostFocus Event Handler

The `amountTextBox_LostFocus` event handler (Fig. 27.18) is called when the `amount-TextBox` *loses the focus* because the user touched another control. In this app, this occurs

when the user touches the `customTipPercentSlider` to change its thumb position. Normally, when a `TextBox` loses the focus, the system *hides* the soft keyboard. In this app, we *force* the keyboard to stay on the screen by calling `amountTextBox`'s `Focus` method when the `TextBox` loses the focus. This allows the user to change the bill amount without having to first touch the `amountTextBox` again. To create this event handler:

1. Select the `amountTextBox` in the **Document Outline** window or in the GUI.
2. In the **Properties** window, click the **Event handlers** icon to display the list of the `TextBox` control's events.
3. Double click the textbox to the right of the `LostFocus` event.
4. Insert the code in line 95.

```

91         // force amountTextBox to keep the focus so keypad remains on screen
92         private void amountTextBox_LostFocus(object sender,
93             RoutedEventArgs e)
94         {
95             amountTextBox.Focus();
96         } // end method amountTextBox_LostFocus
97     } // end class MainPage
98 } // end namespace TipCalculator

```

Fig. 27.18 | `MainPage` class's `amountTextBox_LostFocus` event handler.

27.8 WAppManifest.xml

Each WP8 app project contains the file `WAppManifest.xml`. This so-called *manifest file*—located in the project's **Properties** folder in the **Solution Explorer** window—enables you to set various app settings, such as the name and icons that WP8 displays for your app on a device's **Start** screen, a description of your app, device features the app uses (e.g., device sensors, accessing the user's contacts, network connectivity, etc.), hardware requirements for your app (e.g., gyroscope, camera, etc.) and more. None of these settings were required for you to build and test the **Tip Calculator** app. However, you'll need to specify settings in the manifest file for any app you plan to publish in the Windows Phone Store. For more details on the manifest file, visit:

bit.ly/WAppManifest

27.9 Windows Phone Dev Center

To test your apps on a Windows phone device and to sell your apps or distribute your free apps through the Windows Phone Marketplace, you'll need to join the **Windows Phone Dev Center** (developer.windowsphone.com/) The website includes development tools, sample code, tips for selling your apps, design guidelines, articles, blogs, videos and forums. In this section we provide an overview of the developer registration process and the websites where you'll find step-by-step guidelines.

27.9.1 Microsoft Account

To get started, you must have a *Microsoft account* (formerly a Windows Live ID). If you do not have an account, you can sign up for free at

signup.live.com/signup.aspx?lic=1

27.9.2 Windows Phone Dev Center Account

Next, join the Windows Phone Dev Center at dev.windowsphone.com/en-us/register. There's an annual fee of \$99, payable by credit card or through your PayPal account. The program is *free* to students registered with DreamSpark (www.dreamspark.com/) and to MSDN subscribers. If you're a student but do not have a DreamSpark account, you can sign up for one during the registration process (you must provide proof of your student status). As a registered developer you'll have access to tools, sample code, tips for selling your apps, design guidelines and more.

To register, you must provide your country of residence or business. Registration for publishing and getting paid for apps is limited to developers in certain countries:

bit.ly/WPCountries

You'll also need to select an *account type*—either *company* or *individual/student*. There's a validation process for company accounts. If you're setting up a company account, you'll receive an e-mail from Symantec with a link that you must click to validate the e-mail account. Symantec may request additional documentation. For more information on validating company accounts, see

bit.ly/WPValidateCompanyAcct

Next, you'll enter your contact information and create your unique *Publisher Name* which users will see when they download your app. Companies must use their legal company name, which will be checked in the validation process. Individuals and students may use either their personal name or a “doing business as” name.

Finally, you'll enter either your payment information or DreamSpark student account. Enter your bank account information for electronic deposit or link directly to your PayPal account by providing the mail address associated with it. If you intend to sell your apps or offer in-app purchase, you must also set up a tax profile which contains information such as your social security number or employer tax ID number if you're registering as a corporation. You must also sign the appropriate government tax forms. For more information, see *Getting Paid for Windows Phone* at

bit.ly/WPGettingPaid

For more information about registering for a Windows Phone Dev Center account, see

bit.ly/WPDevCenterAccount

27.9.3 Registering a Windows Phone Device for Development

This chapter shows you how to create a WP8 app and test it on an emulator. If you intend to develop apps for distribution in the Windows Phone Store, you should test your apps

on a Windows Phone device. To learn how to register a device for testing (and how to unregister the device), see *How to register your phone for development* at

bit.ly/WPRegisterPhoneForDev

27.10 Selling Your Apps in the Windows Phone Marketplace

You can sell your own Windows Phone apps in the **Windows Phone Marketplace** (www.windowsphone.com/marketplace), similar to other app commerce platforms such as Apple's App Store, Google Play, Facebook's App Center and the Windows Store. You can also earn money by making your apps free for download and selling *virtual goods* (e.g., additional content, game levels, e-gifts and add-on features) using *in-app purchase*.

27.10.1 Free vs. Paid Apps

A recent study by Gartner found that 89% of all mobile apps are free, and that number is likely to increase to 93% by 2016, at which point in-app purchases will account for over 40% of mobile app revenues.¹ Paid WP8 apps range in price from \$1.49 (which is higher than the \$0.99 starting price for apps in Google Play and Apple's App Store) to \$999.99. The average price for mobile apps is generally between \$1.50 and \$3, depending on the platform. For Windows Phone apps, Microsoft retains 30% of the purchase price and distributes 70% to the app authors. At the time of this writing, there were over 160,000 apps in the Windows Phone Marketplace.²

27.10.2 Submitting Your Apps and In-App Products

Before submitting your apps for publication or **In-App Products** (i.e., virtual goods for in-app purchase), make sure that they comply with Microsoft's requirements for Windows Phone apps (Fig. 27.19). Apps that fail to meet the requirements will be rejected by Microsoft—for example if the app includes illegal content, contains pornography or obscene content, encourages harm or violence to others, prevents the user from disabling location services, sells competing mobile services, contains defamatory or threatening content, infringes on trademarks of others and more. This is similar to Apple's strict approval process for iOS apps and different from Google's Android process where apps are *not* vetted, but they could be removed after the fact.

Document	URL
<i>App certification requirements for Windows Phone</i>	bit.ly/WPAppCertificationRequirements
<i>App policies for Windows Phone</i>	bit.ly/WPAppPolicies

Fig. 27.19 | Microsoft's requirements for WP8 apps. (Part 1 of 2.)

1. techcrunch.com/2012/09/11/free-apps/.
2. thenextweb.com/microsoft/2013/06/27/microsoft-windows-phones-160000-apps-see-more-than-200-million-downloads-monthly/.

Document	URL
<i>Content policies for Windows Phone</i>	bit.ly/WPContentPolicies
<i>App submission requirements for Windows Phone</i>	bit.ly/WPAppSubmissionRequirements
<i>Technical certification requirements for Windows Phone</i>	bit.ly/WPTechnicalCertificationRequirements
<i>Additional requirements for specific app types for Windows Phone</i>	bit.ly/WPAdditionalRequirements

Fig. 27.19 | Microsoft's requirements for WP8 apps. (Part 2 of 2.)

It takes five to seven business days for Microsoft to review a submission for publication in the Windows Phone Store. To learn more about the approval process, see .

bit.ly/WPAppApprovalProcess

27.10.3 Monetizing Apps with Microsoft Advertising pubCenter

Many developers offer free apps monetized with **in-app advertising**—often banner ads similar to those you find on websites. Mobile advertising networks such as **Microsoft Advertising pubCenter** (adsinapps.microsoft.com/pubcenter) aggregate advertisers for you and serve relevant ads within your app on the user's device. You can select the advertising categories that are relevant to your users and the size and placement of ads within the app. You earn advertising revenue based on the number of clickthroughs.

Microsoft Advertising pubCenter is available to registered Windows Phone Developers. To register for a pubCenter account, see pubcenter.microsoft.com/SignUp/WP7.

27.11 Other Popular Mobile App Platforms

According to ABI Research, 56 billion smartphone apps and 14 billion tablet apps will be downloaded in 2013.³ By porting your WP8 apps to other mobile app platforms, especially to Android and iOS (for iPhone, iPad and iPod Touch devices) you could reach an even larger audience (Fig. 27.20). The Windows Phone Dev Center also provides information for porting existing iOS, Android and XAML apps to Windows Phone at

bit.ly/WPPorting

Platform	URL	Worldwide app downloads market share
Android	developer.android.com	58% of smartphone apps 17% of tablet apps

Fig. 27.20 | Popular mobile app platforms. (www.abiresearch.com/press/android-will-account-for-58-of-smartphone-app-down). (Part 1 of 2.)

3. www.abiresearch.com/press/android-will-account-for-58-of-smartphone-app-down.

Platform	URL	Worldwide app downloads market share
iOS (Apple)	developer.apple.com/ios	33% of smartphone apps 75% of tablet apps
Windows Phone 8	developer.windowsphone.com	4% of smartphone apps 2% of tablet apps
BlackBerry	developer.blackberry.com	3% of smartphone apps
Amazon Kindle	developer.amazon.com	4% of tablet apps

Fig. 27.20 | Popular mobile app platforms. (www.abiresearch.com/press/android-will-account-for-58-of-smartphone-app-down). (Part 2 of 2.)

27.12 Developer Documentation

Figure 27.21 lists some of the key Windows Phone developer documentation.

Document	URL
<i>Windows Phone development</i>	bit.ly/WPDevelopment
<i>Visual Studio Express 2012 for Windows Phone</i>	bit.ly/WPVSExpress
<i>Windows Phone Emulator</i>	bit.ly/WPEmulator
<i>How to create your first app for Windows Phone</i>	bit.ly/WPCreateFirstApp
<i>How to create a new app project from a template for Windows Phone</i>	bit.ly/WPNewAppProject
<i>Design library for Windows Phone</i>	bit.ly/WPDesignLibrary
<i>How to deploy and run a Windows Phone app</i>	bit.ly/WPDeployApp
<i>Windows Phone samples: learn through code</i>	bit.ly/WPSampleCode
<i>Testing apps for Windows Phone</i>	bit.ly/WPTestingApps
<i>Windows Phone Store Test Kit</i>	bit.ly/WPStoreTestKit
<i>App certification requirements for Windows Phone</i>	bit.ly/WPAppCertificationRequirements
<i>Monetizing apps for Windows Phone</i>	bit.ly/WPMonetizing

Fig. 27.21 | Some key Microsoft Windows Phone documentation.

27.13 Additional Windows Phone 8 Resources

Microsoft Guides, Blogs and Tutorials

[msdn.microsoft.com/en-us/library/windowsphone/develop/ff402529\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402529(v=vs.105).aspx)

The Microsoft guide, *Getting started with developing for Windows Phone*, discusses building your first app and testing it on a device.

[msdn.microsoft.com/en-us/library/windowsphone/develop/ff402526\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402526(v=vs.105).aspx)

The Microsoft guide, *How to create your first app for Windows Phone*, walks you through the development of a simple web browser app.

[msdn.microsoft.com/en-us/library/windowsphone/develop/jj206958\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206958(v=vs.105).aspx)

The Microsoft Guide, *Speech for Windows Phone 8*, discusses voice commands, speech recognition and text-to-speech and handling errors in speech apps for WP8.

blogs.msdn.com/b/matthiasshapiro/archive/2013/03/21/10-great-windows-phone-8-code-samples-no-wait-30.aspx

The MSDN blog, “10 Great Windows Phone 8 Code Samples,” by Matthias Shapiro, highlights 10 of the authors favorite sample apps from the Microsoft Windows Phone 8 Samples collection.

[msdn.microsoft.com/en-US/library/windowsphone/develop/hh394031\(v=vs.105\).aspx](http://msdn.microsoft.com/en-US/library/windowsphone/develop/hh394031(v=vs.105).aspx)

The guide, *Porting your app to Windows Phone*, from the Windows Phone Dev Center, provides instructions and API mapping tools for porting existing XAML, iOS and Android apps to Windows Phone.

channel9.msdn.com/Series/Building-Apps-for-Windows-Phone-8-Jump-Start/Building-Apps-for-Windows-Phone-8-Jump-Start-01a-Introducing-Windows-Phone-8-Development-Part-1

The video tutorial, “Building Apps for Windows Phone 8 Jump Start: Introducing Windows Phone 8 Development Part 1,” by Andy Wigley and Rob Tiffany. Topics include WP8 APIs and app development models. The second part of the tutorial discusses designing and building apps, app life-cycle, tiles and lock screen notifications and more.

Other Online Tutorials

www.youtube.com/watch?v=b1U9rBiVURM

The free video tutorial, “Windows Phone 8 Development—Tutorial 1—New Application.”

codeproject.tv/video/5036055/setting_up_a_command_mvvm_in_windows_phone_8

The video tutorial, “Setting up a Command (Model View View-Model) in Windows Phone 8,” by Michael Crump—a Microsoft MVP.

www.codeproject.com/Articles/521530/A-Windows-Phone-8-Run-Tracking-App-in-100-Lines-of

The tutorial, “A Windows Phone 8 Run Tracking App in 100 Lines of Code,” by Colin Eberhardt. Topics include the new WP8 map control, tracking the user’s location, using live tiles and more.

www.lynda.com/Visual-Studio-tutorials/Up-Running-Windows-Phone-8-Development/117545-2.html

The fee-based online training course, “Up and Running with Windows Phone 8 Development,” by Doug Winnie. This course teaches introductory WP8 app development. You’ll learn how to set up the Visual Studio tools, build a simple tip calculator app in C# and XAML and how to submit your app to the Windows Store.

Nokia Windows Phone Resources

www.developer.nokia.com/Develop/Windows_Phone/Code_examples/

Nokia’s Windows Phone code examples site includes samples of several fully implemented WP8 apps.

www.developer.nokia.com/Community/Wiki/Speech_Enabled_Calculator_For_Windows_Phone_8

The tutorial, “Speech Enabled Calculator For Windows Phone 8.” This app uses several APIs including Voice Commands, Speech Recognition and Text To Speech. The tutorial walks you through the development of a calculator app that accepts input and provides output via both touch and speech commands.

Open-Source Windows Phone Tools

coding4fun.codeplex.com

Coding4Fun Toolkit includes controls, audio, network and storage wrappers for XAML based apps.

phone.codeplex.com

Windows Phone Toolkit from Microsoft's Windows Phone developer platform team includes components for building WP8 apps. The site also includes open source code samples.

wptools.codeplex.com

Windows Phone Power Tools—Includes tools for testing updates to existing apps, and a GUI for interacting with your apps.

27.14 Wrap-Up

In this chapter, you created an interactive WP8 app—the **Tip Calculator**. We overviewed the app's capabilities, then you test-drove it to calculate tips based on the bill amount and tip percentage entered. You followed detailed step-by-step instructions to build the app's GUI using the Visual Studio Express 2012 for Windows Phone IDE. In the app's GUI, you used a **Grid** to arrange the controls into rows and columns. You displayed text in **TextBlocks** and received input from a **TextBox** and a **Slider**. Using the IDE's GUI designer, **Document Outline** window and **Properties** window, you built the app's GUI without manipulating XAML directly.

We also walked through the code of the **PhoneApplicationPage** derived class **MainPage**, which defines the app's functionality. You responded to its **Loaded** event to customize various controls after they've been created. You handled a **TextBox**'s **TextChanged** event so the app could calculate new tips and totals as the user changed the text in the **TextBox**. You also handled a **Slider**'s **ValueChanged** event so the app could calculate a new custom tip and total as the user changed the custom tip percentage by moving the **Slider**'s *thumb*.

We walked through the registration process for Windows Phone Dev Center so you can test your WP8 apps and distribute them free or for a fee in the Windows Phone Store. We discussed how to prepare apps and in-app products for submission to the Windows Phone Store, including testing them on the emulator and following Microsoft's requirements for Windows Phone apps. We discussed pricing your apps, and resources for monetizing them with in-app advertising and in-app products.

Exercises

27.1 (*Enhanced Tip Calculator App*) Make the following enhancements to the **Tip Calculator** app:

- Add an option to calculate the tip based on either the price before tax or after tax.
- Allow the user to enter the number of people in the party. Calculate and display the amount owed by each person if the bill were to be split evenly among the party members.

27.2 (*Mortgage Calculator App*) Create a mortgage calculator app that allows the user to enter a purchase price, down payment amount and an interest rate. Based on these values, the app should calculate the loan amount (purchase price minus down payment) and display the monthly payment for 10, 20 and 30 year loans. Allow the user to select a custom loan duration (in years) by using a **Slider** and display the monthly payment for that custom loan duration.

27.3 (College Loan Payoff Calculator App) A bank offers college loans that can be repaid in 5, 10, 15, 20, 25 or 30 years. Write an app that allows the user to enter the amount of the loan and the annual interest rate. Based on these values, the app should display the loan lengths in years and their corresponding monthly payments.

27.4 (Car Payment Calculator App) Typically, banks offer car loans for periods ranging from two to five years (24 to 60 months). Borrowers repay the loans in monthly installments. The amount of each monthly payment is based on the length of the loan, the amount borrowed and the interest rate. Create an app that allows the customer to enter the price of a car, the down-payment amount and the loan's annual interest rate. The app should display the loan's duration in months and the monthly payments for two-, three-, four- and five-year loans. The variety of options allows the user to easily compare repayment plans and choose the most appropriate.

27.5 (Miles-Per-Gallon Calculator App) Drivers often want to know the miles per gallon their cars get so they can estimate gasoline costs. Develop an app that allows the user to input the number of miles driven and the number of gallons used and calculates and displays the corresponding miles per gallon.

27.6 (Body Mass Index Calculator App) The formulas for calculating the BMI are

$$BMI = \frac{weightInPounds \times 703}{heightInInches \times heightInInches}$$

or

$$BMI = \frac{weightInKilograms}{heightInMeters \times heightInMeters}$$

Create a BMI calculator app that allows users to enter their weight and height and whether they are entering these values in English or Metric units, then calculates and displays the user's body mass index. The app should also display the following information from the Department of Health and Human Services/National Institutes of Health so the user can evaluate his/her BMI:

BMI VALUES

Underweight: less than 18.5

Normal: between 18.5 and 24.9

Overweight: between 25 and 29.9

Obese: 30 or greater

27.7 (Target-Heart-Rate Calculator App) While exercising, you can use a heart-rate monitor to see that your heart rate stays within a safe range suggested by your trainers and doctors. According to the American Heart Association (AHA), the formula for calculating your *maximum heart rate* in beats per minute is *220 minus your age in years* (<http://bit.ly/AHATargetHeartRates>). Your *target heart rate* is a range that is 50–85% of your maximum heart rate. [Note: These formulas are estimates provided by the AHA. Maximum and target heart rates may vary based on the health, fitness and gender of the individual. Always consult a physician or qualified health care professional before beginning or modifying an exercise program.] Write an app that inputs the person's age, then calculates and displays the person's maximum heart rate and target-heart-rate range.

