Windows 10

Hands-on lab

Lab: Hello World Across UWP Devices

August 2015

Microsoft

CONTENTS

Overview

Windows 10 introduces the Universal Windows Platform (UWP), which further evolves the Windows Runtime model and brings it into the Windows 10 unified core. As part of the core, the UWP now provides a common app platform available on every device that runs Windows 10. With this evolution, apps that target the UWP can call APIs specific to the device family in addition to the WinRT APIs that are common to all devices. The UWP provides a guaranteed core API layer across devices. With UWP, you can create a single app package that can be installed onto a wide range of devices.

In this lab, you will use the Universal Windows App Development Tools to build a Hello World app that runs on all Windows 10 devices. Your app will display information about the device it is running on, including the device family and current app window size. You will also create a Hello World app directly from Blend and leverage Blend's ability to generate sample data.

Objectives

This lab will show you how to:

- Create a Universal Windows app from the Blank App template
- Display a greeting in your app
- Detect and display the device family
- Dynamically display the app window size
- Deploy to the Local Machine
- Deploy to the Mobile emulator
- Deploy to an IoT device
- Generate sample data in Blend

System requirements

You must have the following to complete this lab:

- Microsoft Windows 10
- Microsoft Visual Studio 2015
- Windows 10 Mobile Emulator

Optional add-ons

If you wish to complete the optional tasks in this lab, you will need:

- An IoT device running Windows 10
- A display that connects to the IoT device

Setup

You must perform the following steps to prepare your computer for this lab:

1. Install Microsoft Windows 10.
2. Install Microsoft Visual Studio 2015. Choose a custom install and ensure that the Universal Windows App Development Tools are selected from the optional features list.
3. Install the Windows 10 Mobile Emulator.
4. Optional: Install Windows 10 on an IoT device.

Exercises

This Hands-on lab includes the following exercises:

1. Getting started with UWP
2. Hello World Across Devices
3. Hello World in Blend

Estimated time to complete this lab: **30 to 45 minutes**.

Exercise 1: Getting Started with UWP

The Universal Windows App Development Tools provides a template to help you get started building your own UWP apps. In this exercise, you will create a project from the Blank App template and explore what it has to offer.

Task 1 – Create a blank Universal Windows app

We will begin by creating a project from the Blank App template.

1. In a new instance of Visual Studio 2015, choose File > New> Project to open the New Project dialog. Navigate to Installed > Templates > Visual C# > Windows > Universal and select the Blank App (Universal Windows) template.

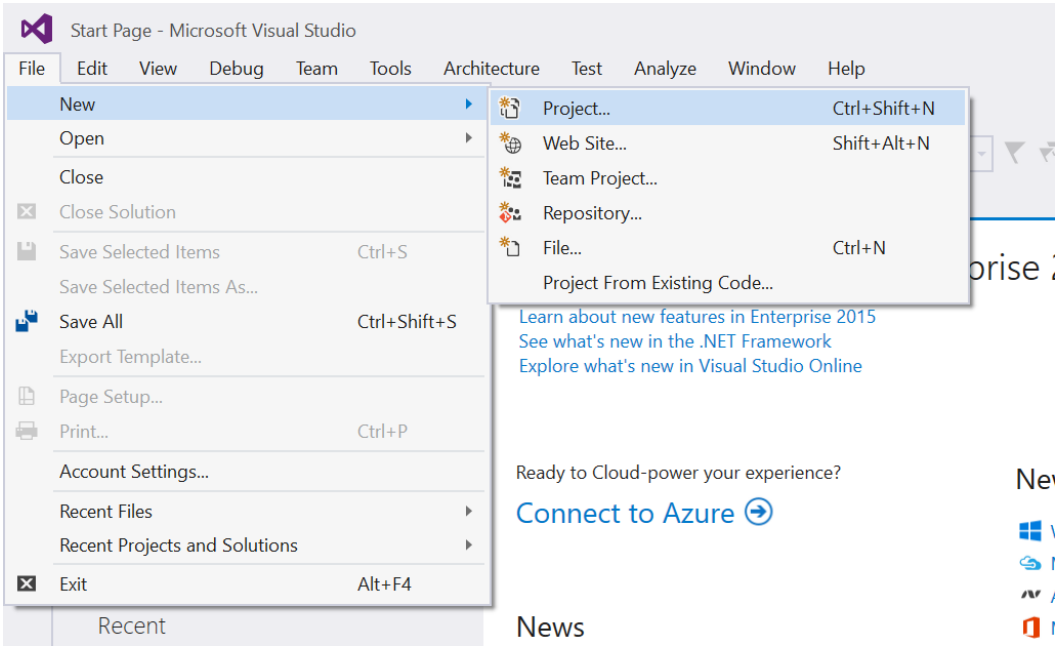

Figure 1

Open the New Project dialog in Visual Studio 2015.

2. Name your project **HelloUWP** and select the filesystem location where you will save your Hands-on Lab solutions. We have created a folder in our **C:** directory called **HOL** that you will see referenced in screenshots throughout the labs.

3. Leave the options selected to **Create new solution** and **Create directory for solution**. You may deselect **Add to source control** if you don't wish to version your work. Click **OK** to create the project.

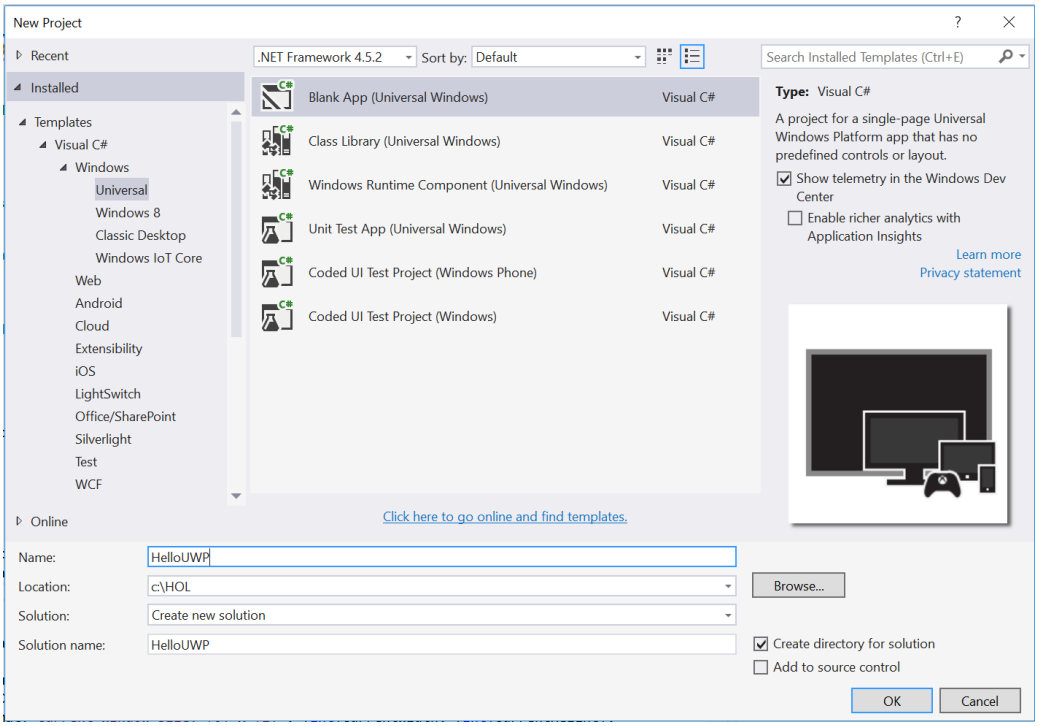

Figure 2

Create a new Blank App project in Visual Studio 2015.

4. Set your Solution Configuration to Debug and your Solution Platform to x86. Select Local Machine from the Debug Target dropdown next to the Start Debugging Button.

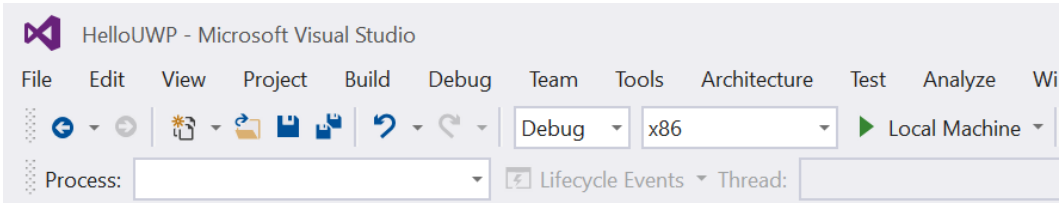

Figure 3

Configure your app to run on the Local Machine.

Note: ▶ is the Start Debugging button.

5. Click the Start Debugging button to build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.

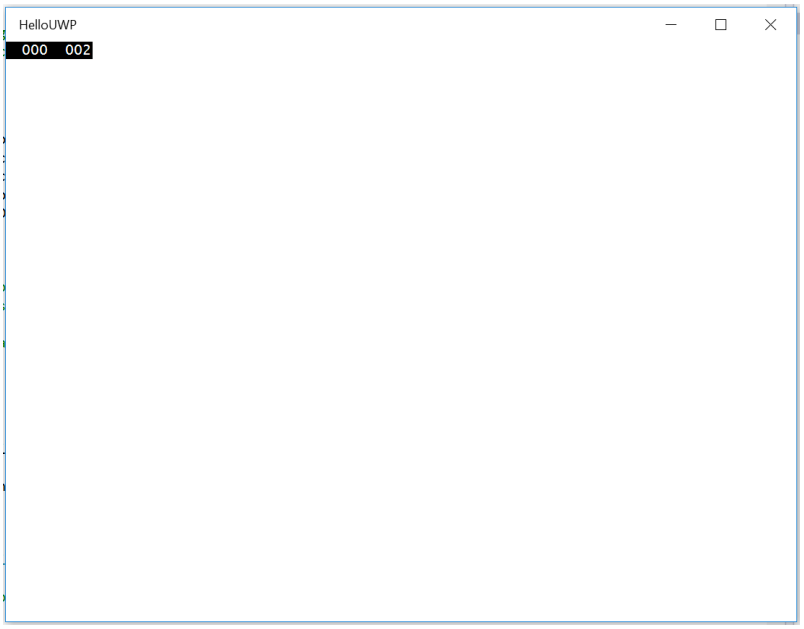

Figure 4

The Blank universal app running in Desktop mode.

Note: The frame rate counter is a debug tool that helps to monitor the performance of your app. It is useful for apps that require intensive graphics processing but unnecessary for the simple apps you will be creating in the Hands-on Labs.

In the Blank App template, the preprocessor directive to enable or disable the frame rate counter is in **App.xaml.cs**. The frame rate counter may overlap or hide your app content if you leave it on. For the purposes of the Hands-on Labs, we will turn it off.

6. Return to Visual Studio and stop debugging.

7. Open App.xaml.cs. Turn off the frame rate counter in the **#if DEBUG** preprocessor directive by setting **this.DebugSettings.EnableFrameRateCounter** to **false**.

C#

```
#if DEBUG
    if (System.Diagnostics.Debugger.IsAttached)
    {
        this.DebugSettings.EnableFrameRateCounter = false;
    }
#endif
```

8. Build and run your app again. This time, you will see a blank app window without the frame rate counter.

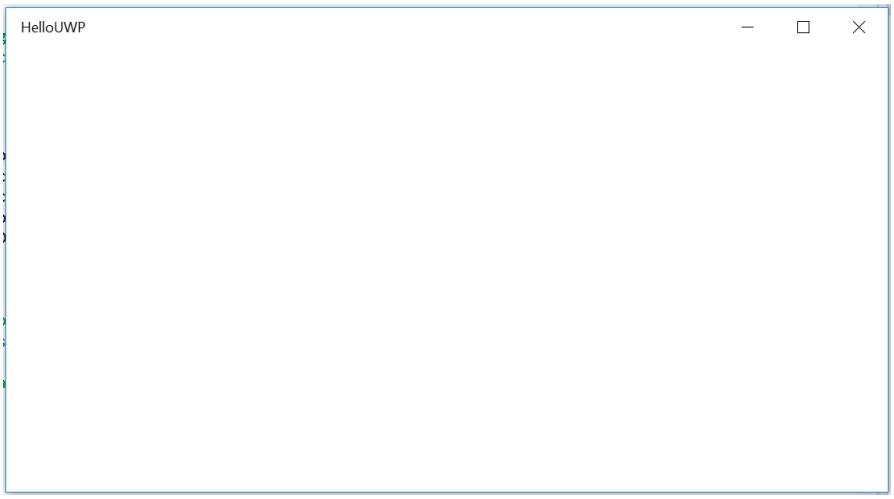

Figure 5

The Blank universal app after the frame rate counter has been disabled.

9. Stop debugging and return to Visual Studio.

Task 2 - Explore the template

The Blank App template provides the basic structure you will need to create a UWP app. Let's take a look at some of its features.

1. **App.xaml.cs** is the entry point for your app and contains methods to handle activation and suspension. In this template, its constructor initializes Application Insights, calls InitializeComponent(), and listens for the Suspending event. In the corresponding **App.xaml**, you can declare resources that will be shared across the app.

   The **OnLaunched** override in App.xaml.cs sets up the root frame as the navigation context and attaches it to the window. You may add code to load state from a previous execution; if no state is restored, the frame will navigate to the **MainPage** view by default.

   You may also notice that the **OnLaunched()** override includes code to enable the frame rate counter when debugging, conditionally included by the `#if` DEBUG preprocessor directive. You can learn more about some of the debug settings available to you at https://msdn.microsoft.com/en-us/library/windows.ui.xaml.debugsettings.aspx.

   Window.Current.Activate() is an important part of the activation process and is required for all app activation scenarios, including secondary windows.

2. **MainPage.xaml** and **MainPage.xaml.cs** together define the **MainPage** view, which inherits from the **Page** class. You can add UI elements and layout to **MainPage.xaml**, and add logic and event handlers to the **MainPage.xaml.cs** code behind. You will learn more about customizing the MainPage view in the next exercise.

3. The **Assets** folder contains the default assets for the app. In the Blank App template, these assets are minimal. If you wish to provide custom or additional assets, you may add them to this directory by right-clicking on the folder name in the Solution Explorer and selecting **Add > Existing Item**. Once they have been added, you can visit the app manifest to register your new assets.

4. The **Package.appxmanifest** file describes and identifies your app. Here you can add capabilities and extensions – for instance, the device capability to access the microphone – and reference your splash screen, default tile, and logo assets. **Package.appxmanifest** opens by default in the **Manifest editor**, which provides useful information that may not be apparent when viewing the code directly. On the **Visual Assets** tab, for instance, the project editor indicates which logo assets are recommended. You may still view the app manifest as XML: to do so, right-click on the file in the Solution Explorer and choose **View Code**.

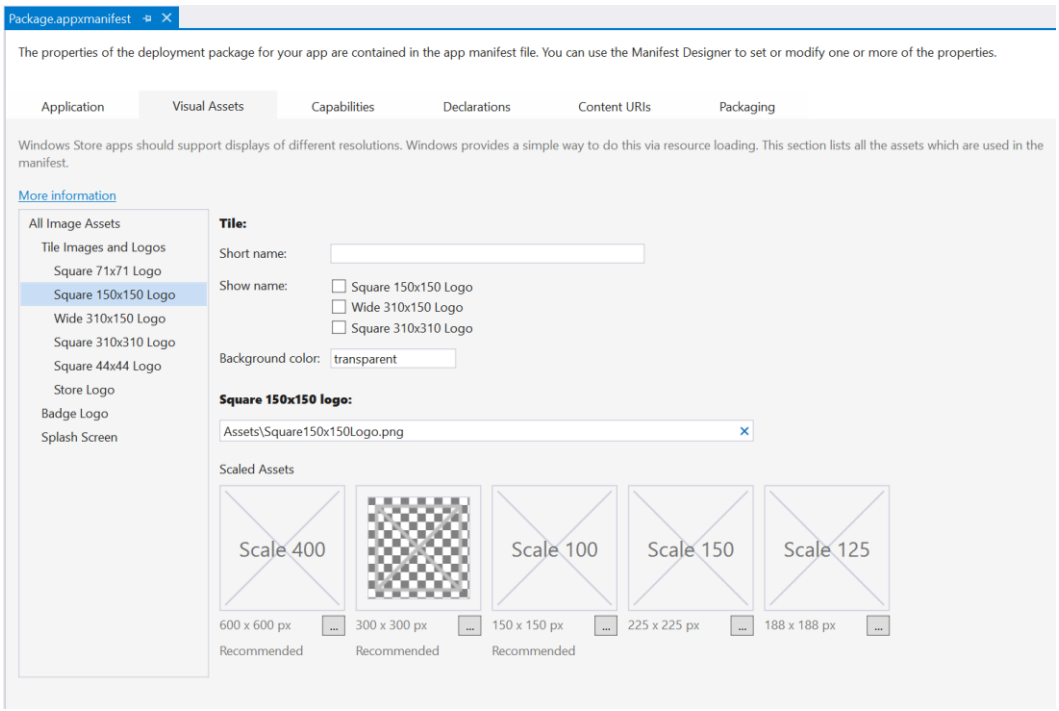

Figure 6

The app manifest indicates which assets are recommended when viewed in the Manifest editor.

5. The **References** folder lists SDKs, libraries, and components that are available to your app. If you wish to add a reference, right-click on the References folder and choose **Add Reference** to open the **Reference Manager**.

6. The **project.json** file in the root directory of your project is new in UWP and simplifies management of NuGet packages. You do not normally edit this file directly, but instead use the

NuGet Package manager tool to discover and manage packages to use in your project. For more information, visit http://docs.nuget.org/consume/ProjectJson-Intro.

7. The **Properties** file allows you to manage build events and your debug configuration. As Windows 10 evolves, you may wish to visit the **Application** tab to update the target version and min version of Windows 10 that your app can support.

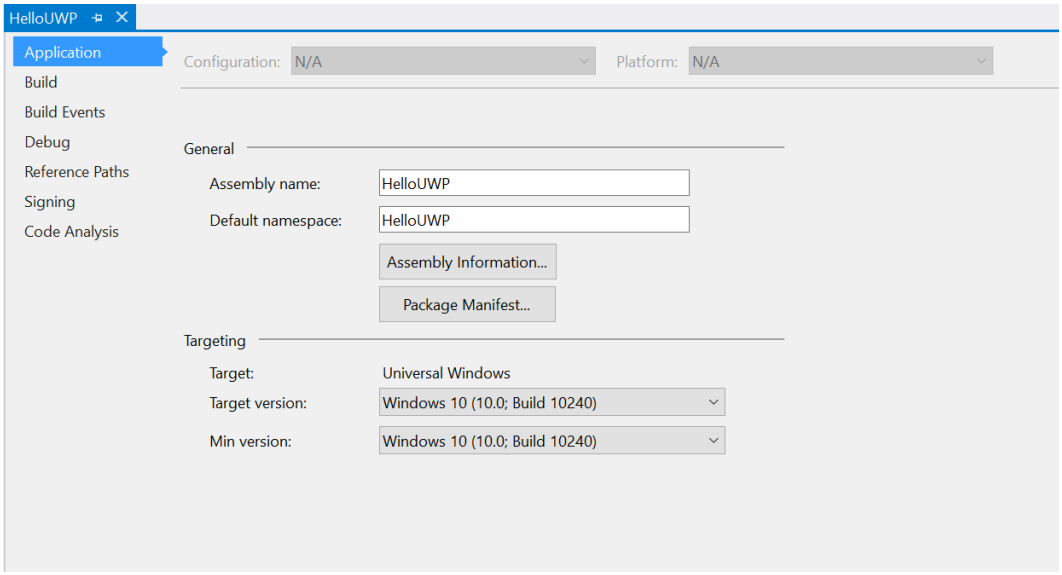

Figure 7

Change your supported target version and min version in the Properties for your app.

Exercise 2: Hello World Across Devices

Windows Apps are now universal across devices, which means that they can run in full screen mobile and tablet modes as well as windowed desktop mode. In this exercise, you will create a Hello World app that displays a greeting and device information across all Windows 10 devices, including desktop, tablet, mobile, and IoT.

Task 1 – Display a greeting

The first task is to open the HelloUWP solution you created in the previous exercise.

1. Navigate to the file location where you saved your **HelloUWP** app in Exercise 1. Open **HelloUWP.sln** in Visual Studio 2015. Alternatively, you may open the solution using the Visual Studio **Open Project** dialog.

2. Open **MainPage.xaml** from the Solution Explorer. Add a **StackPanel** containing a **TextBlock** to the existing grid.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <TextBlock />
    </StackPanel>
</Grid>
```

Note: The **StackPanel** is a control that stacks child elements vertically or horizontally. There are a number of other container controls that can assist in the layout of your UI, including **Grid**, **Canvas**, and **RelativePanel**. We will cover the Grid and RelativePanel controls in more depth in the **Building an Adaptive UI** lab.

3. Add text and styling to your TextBlock.

XAML

```
<TextBlock Text="Hello UWP World!" FontSize="24" FontWeight="Light"
Margin="12" />
```

Note: You may notice throughout the Hands-on Labs that our margin, size, and position values are always multiples of 4. UWP apps use effective pixels on a 4x4 pixel grid to ensure a high quality visual experience across devices using screens at different resolutions and scale factors. To ensure a crisp appearance for your app, snap your UI elements to the 4x4 pixel grid. The grid does not apply to text, however, so you may continue to use any font size you wish.

For more on effective pixels and responsive design, visit https://msdn.microsoft.com/en-us/library/windows/apps/Dn958435.aspx

4. Build and run your app on the **Local Machine**.

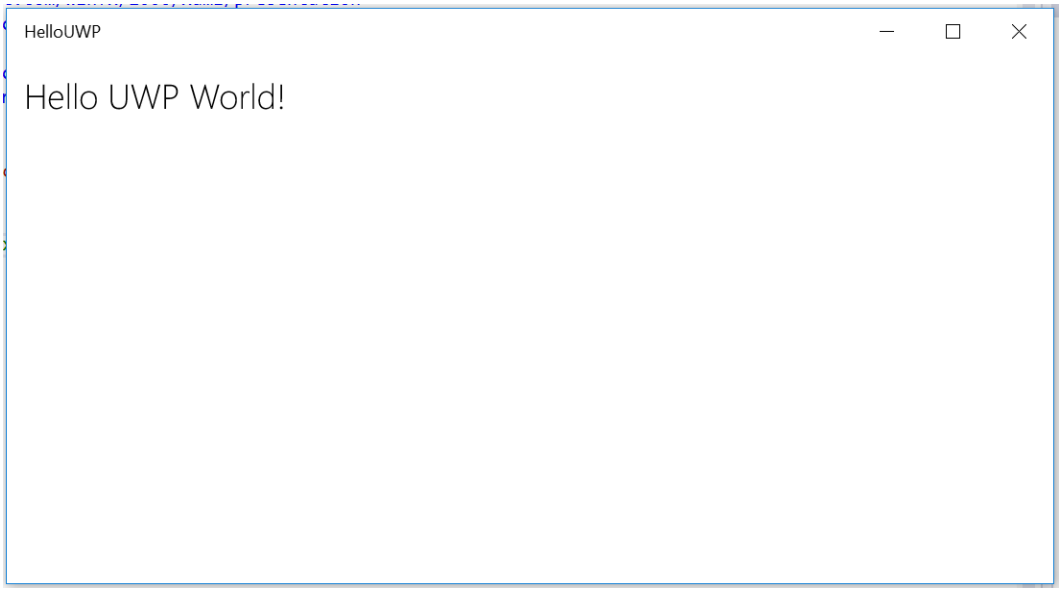

Figure 8

The HelloUWP app displays a greeting.

5. Stop debugging and return to Visual Studio. Later in this exercise, you will learn how to deploy to the mobile emulator and IoT devices. First, you will add elements to display information that will create a richer experience on those devices.

Task 2 - Detect the device family

Now that a single UWP app runs across all Windows 10 devices, it would be useful to have the ability to target families of devices that share features and APIs. You may wish to display certain features that are available on mobile, for instance, but hide those features on Desktop. To target a device, we must first detect the device family to which it belongs. In this task, you will detect and display the device family on which your app is running.

1. Open **MainPage.xaml.cs**. Add a field to access the device family.

C#

```
public string DeviceFamily = "Device Family " +
                             AnalyticsInfo.VersionInfo.DeviceFamily;

public MainPage()
{
    this.InitializeComponent();
}
```

Note: The way of thinking about app targeting changes with Windows 10. The new conceptual model is that an app targets a group of related devices called a device family. A device family is a set of APIs

collected together and given a name and a version number. Your app can run on a variety of device families while targeting specific features unique to each one through adaptive code.

For more on device families in UWP apps, visit https://msdn.microsoft.com/en-us/library/windows/apps/dn894631.aspx

2. Add the **Windows.System.Profile** namespace to the MainPage codebehind.

C#

```
using Windows.System.Profile;
```

3. In MainPage.xaml, bind a **TextBlock** to the **DeviceFamily** field.

Note: Data binding provides a simple way for your UI to interact with data. The display of data is de-coupled from the business logic for the app, and the binding allows properties of UI elements to connect with data fields. Binding can be used for a one-time display of data, updating the UI when the data changes, or for capturing user input.

{x:Bind} is a new markup extension in Windows 10 UWP and offers a compiled alternative to the classic {Binding}. For more on x:Bind, visit https://msdn.microsoft.com/en-us/library/windows/apps/Mt204783.aspx

We will cover binding in the **Data Binding** lab.

XAML

```
<StackPanel>
    <TextBlock Text="Hello UWP World!" FontSize="24" FontWeight="Light"
Margin="12" />
    <TextBlock Text="{x:Bind DeviceFamily}" Margin="12,0,0,0" />
</StackPanel>
```

Note: The StackPanel control stacks items vertically by default. You can specify horizontal orientation with the attribute Orientation=”Horizontal”.

4. Build and run your app on the **Local Machine**. You will see the device family display below the greeting.

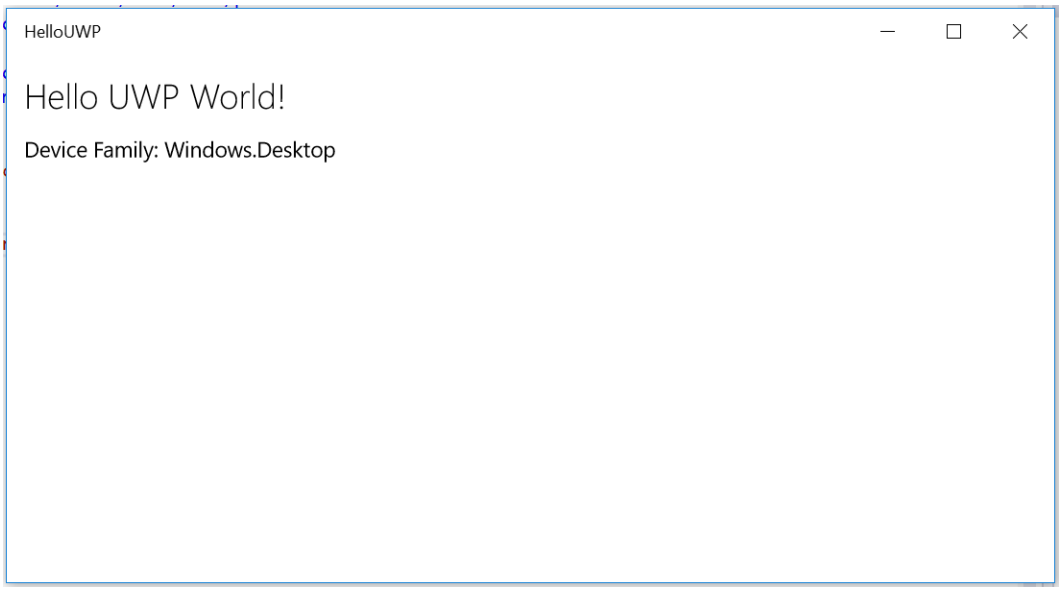

Figure 9

The HelloUWP app displays the device family of the device on which it is currently running.

Note: If you are developing in a desktop environment and deploy to the Local Machine, your device family will display as Windows.Desktop. Additional device families available in the Universal Device Family include Mobile, Xbox, IoT, and IoT headless.

5. Stop debugging and return to Visual Studio.

Task 3 – Dynamically show the dimensions of the app window

Another property that is interesting to compare across devices is app window size, which is expressed in effective pixels. Universal Windows apps can run in full screen mobile and tablet modes as well as windowed desktop mode. In this task, you will create a read-out to display the current app window dimensions. The display will update when you resize the app window.

1. Add a **TextBlock** to your **StackPanel** in MainPage.xaml and bind it to a **Dimensions** property. You will create the **Dimensions** property in later steps. Note that with x:Bind the default binding mode is OneTime, so for the TextBlock to update whenever the Dimensions property changes, you must specify the binding Mode as **OneWay**.

XAML

```
<StackPanel>
    <TextBlock Text="Hello UWP World!" FontSize="24" FontWeight="Light"
Margin="12" />
    <TextBlock Text="{x:Bind DeviceFamily}" Margin="12,12,0,0" />
    <TextBlock Text="{x:Bind Dimensions, Mode=OneWay}" Margin="12,0,0,0" />
</StackPanel>
```

1. Open **MainPage.xaml.cs** and create properties for dimensions, current width, and current height.

C#

```
public string DeviceFamily = "Device Family " +
                             AnalyticsInfo.VersionInfo.DeviceFamily;

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

2. Add **a MainPage_SizeChanged** routed event handler to the code behind. When the **SizeChanged** event fires, this handler will ascertain the new width and height of the window and round them to integer values for display.

C#

```
public MainPage()
{
    this.InitializeComponent();
    this.SizeChanged += MainPage_SizeChanged;
}

private void MainPage_SizeChanged(object sender, SizeChangedEventArgs e)
{
    var currentWidth = Window.Current.Bounds.Width;
    var currentHeight = Window.Current.Bounds.Height;
    Dimensions = string.Format("Current Window Size: {0} x {1}",
                               (int)currentWidth, (int)currentHeight);
}
```

3. For your binding to work to update the UI when the **Dimensions** property changes, you must raise a **PropertyChanged** event. Add the **System.ComponentModel** namespace to the MainPage code-behind which is the namespace for the **INotifyPropertyChanged** interface.

C#

```
using System.ComponentModel;
```

4. Implement the **INotifyPropertyChanged** interface.

C#

```
public sealed partial class MainPage : Page, INotifyPropertyChanged
{
    public string DeviceFamily = "Device Family " +
                             AnalyticsInfo.VersionInfo.DeviceFamily;

    public event PropertyChangedEventHandler PropertyChanged;

    public string Dimensions { get; set; } = "Initial Value";
```

```
    private double _currentWidth;
    private double _currentHeight;


    public MainPage()
    {
        this.InitializeComponent();
        this.SizeChanged += MainPage_SizeChanged;
    }

    private void MainPage_SizeChanged(object sender, SizeChangedEventArgs e)
    {
        _currentWidth = Window.Current.Bounds.Width;
        _currentHeight = Window.Current.Bounds.Height;
        Dimensions = string.Format("Current Window Size: {0} x {1}",
(int)_currentWidth, (int)_currentHeight);

        if (PropertyChanged != null)
        {
            PropertyChanged(this,
                            new PropertyChangedEventArgs(nameof(Dimensions)));
        }
    }
```

5. Build and run your app. The **Dimensions** TextBlock will display the initial window size and will update as you resize the window.

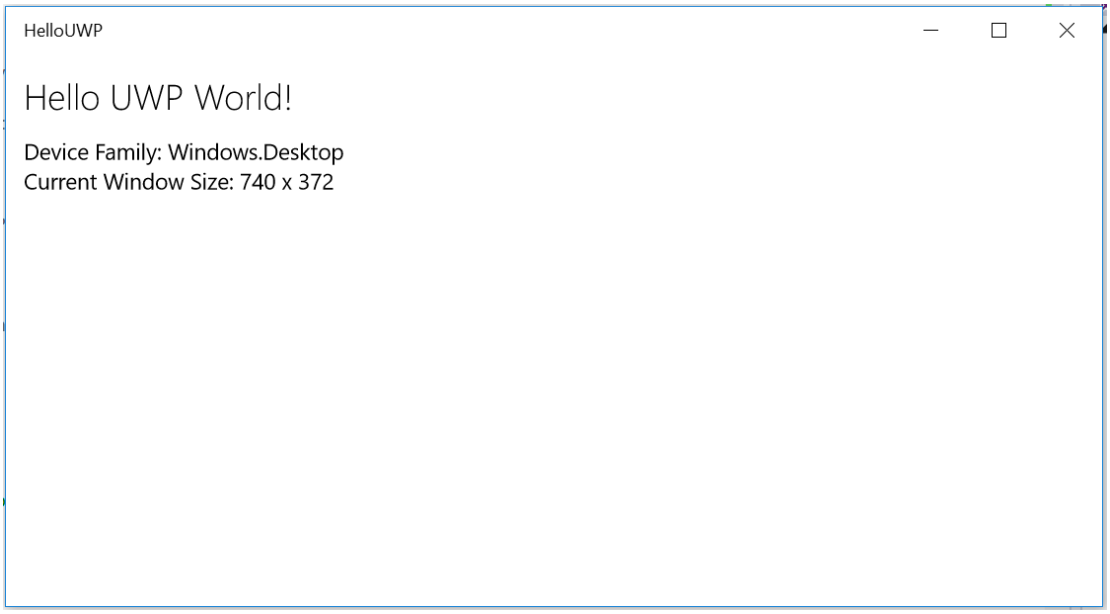

Figure 10

The dimensions update as the window is resized.

6. Stop debugging and return to Visual Studio.

Task 4 - Deploy to the Mobile emulator

The HelloUWP app you created in the previous task is already set for deployment to any Windows 10 device. Prior to UWP app development, you would have had to create a separate project to deploy to mobile devices. The beauty of the UWP is that a single project will run on all your Windows 10 devices. In this task, we will use the Windows 10 Mobile emulator to experience the HelloUWP app on mobile.

Note: Microsoft Emulator for Windows 10 Mobile ships as part of the Windows 10 SDK and can be installed during the Visual Studio 2015 installation process.

If you did not install the emulator with Visual Studio 2015, you can install it using the Microsoft Emulator setup. Visit the Windows 10 Tools download page for more information.

1. Change your **Debug Target** to use one of the mobile emulators provided by the Microsoft Emulator for Windows 10 Mobile. We have chosen to use the **Mobile Emulator 10.0.10240.0 720p 5 inch 1GB** option.

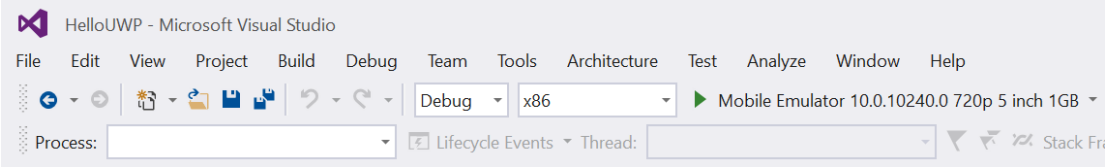

Figure 11
The debug target is set to a Mobile Emulator.

2. Build and run your app. The HelloUWP app will display on your mobile device with the Windows.Mobile device family. The current window size will reflect the screen dimensions of the mobile device (with the exception of the status bar), because mobile apps always run in fullscreen mode.

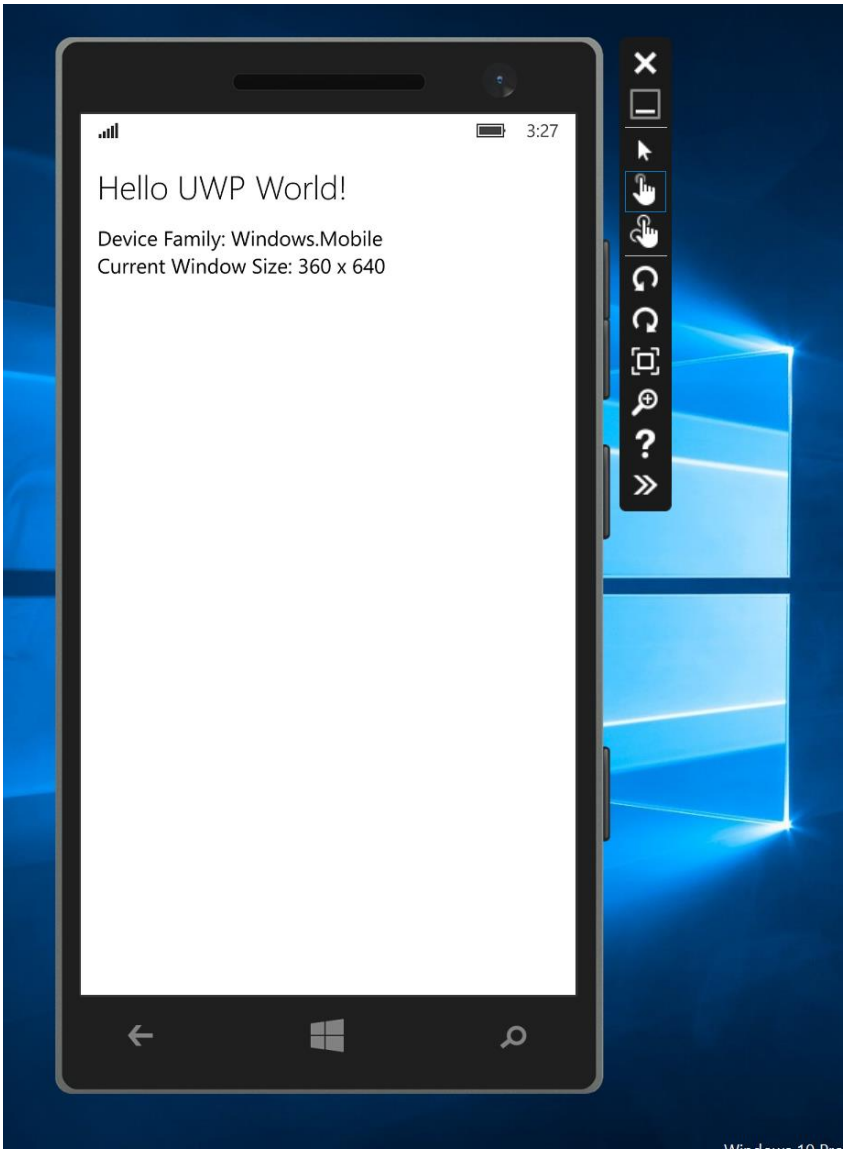

Figure 12

The HelloUWP runs on a Mobile Emulator.

3. Click on one of the rotate buttons on the emulator toolbar to change to landscape mode. Your app will update its window size information.

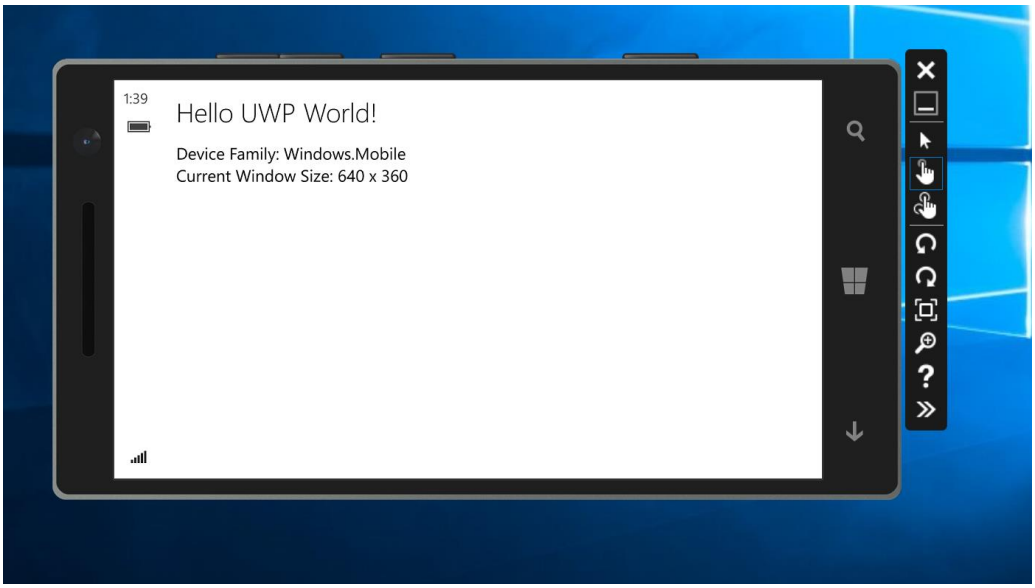

Figure 13
The window size display updates in landscape mode.

4. Close the emulator. Stop debugging and return to Visual Studio.

Task 5 - Deploy to IoT devices (optional)

The Windows 10 Universal device family includes the Internet of Things (IoT) device family in addition to Desktop, Mobile, and Xbox. IoT devices typically combine sensors and connectivity to record and exchange data with other devices. You may follow this task to deploy your HelloUWP app to an IoT device if you have one available. For this demo, we will use a Raspberry Pi 2 running Windows 10. If you do not have access to an IoT device, you may wish to skip this task.

Note: Visit **http://www.windowsondevices.com/** for instructions and tools to install Windows 10 on an IoT device. Supported devices include the Raspberry Pi, Minnowboard Max, Galileo, and Arduino.

The Universal device family also includes the IoT headless device family. Headless IoT devices operate without a graphical user interface. Our HelloUWP app displays information visually, so we will deploy to the IoT family instead of the IoT headless family.

1. Make sure your IoT device has power and boot it into Windows 10. Connect the device to a display.

2. Connect your IoT device to your local network. You may connect directly via ethernet or Wi-Fi or use Internet Connection Sharing (ICS) to connect through your development machine.

Note: For more information on connecting your IoT device to your local network, visit https://ms-iot.github.io/content/en-US/win10/ConnectToDevice.htm

3. Determine the local IP address of your IoT device. A Raspberry Pi 2 running the default Windows 10 for IoT devices displays its device name and IP address on the home screen.

4. Use Powershell to connect and configure your Windows 10 IoT Core device as described here: http://ms-iot.github.io/content/en-US/win10/samples/PowerShell.htm

5. Open the HelloUWP project in Visual Studio on your development machine. Set the Solution Platform to **ARM**.

6. Double-click on the project **Properties** in the Solution Explorer. Alternatively, you may right-click on the project name and select **Properties** from the menu.

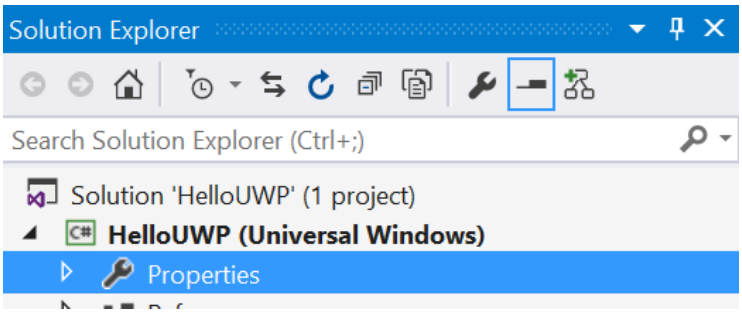

Figure 14

Open the project properties from the Solution Explorer.

1. Navigate to the **Debug** properties tab. Use the **Target device** dropdown to select **Remote Machine** as your debug target.

2. Deselect the **Use authentication** option.

3. Type your IoT device name or local IP address into the **Remote machine** field.

4. Save the HelloUWP properties file.

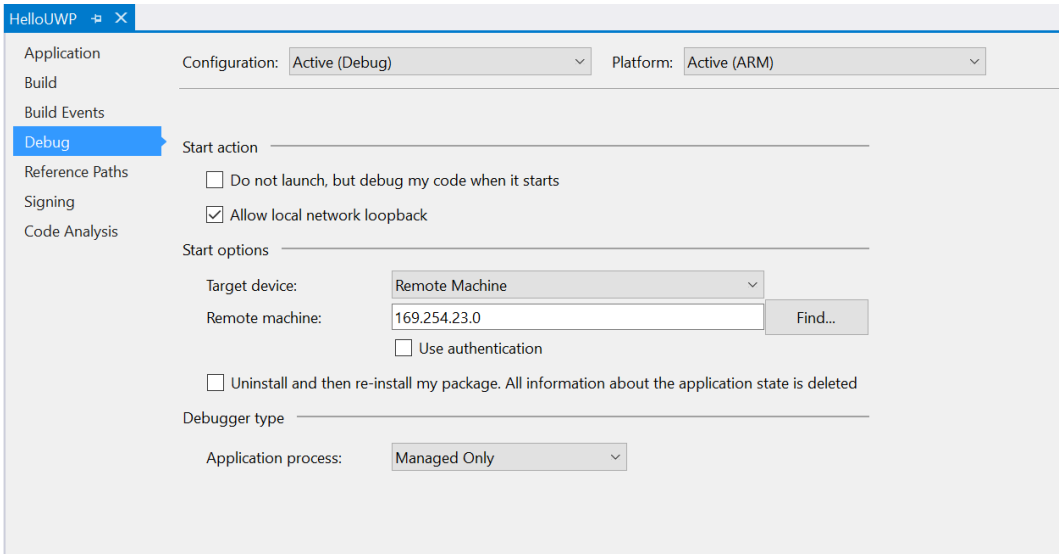

Figure 15

Configure your project properties to deploy to a remote machine on the local network.

5. Build and run your app. You will see it launch on the display connected to your IoT. The Device Family will show as **Windows.IoT**.

6. Stop debugging and return to Visual Studio.

Exercise 3: Hello World in Blend

Blend for Visual Studio 2015 has been redesigned to make it even easier to get started designing and building your app. In addition to UI and workflow improvements, XAML IntelliSense and basic debugging capabilities are now supported. In this exercise, we will use Blend to create a new project and populate it with sample data.

Task 1 – Create a new project in Blend

Blend gives you the ability to start a building a new app without opening Visual Studio.

1. Launch Blend for Visual Studio 2015. Use the **Start Page** pane or **File > New > Project** to open the **New Project** dialog.

2. Under **Templates > Visual C#**, select the **Blank App (Universal Windows) template.**

3. Give your project the name **HelloBlend** and save it to the folder where you store your Hands-on Labs projects.

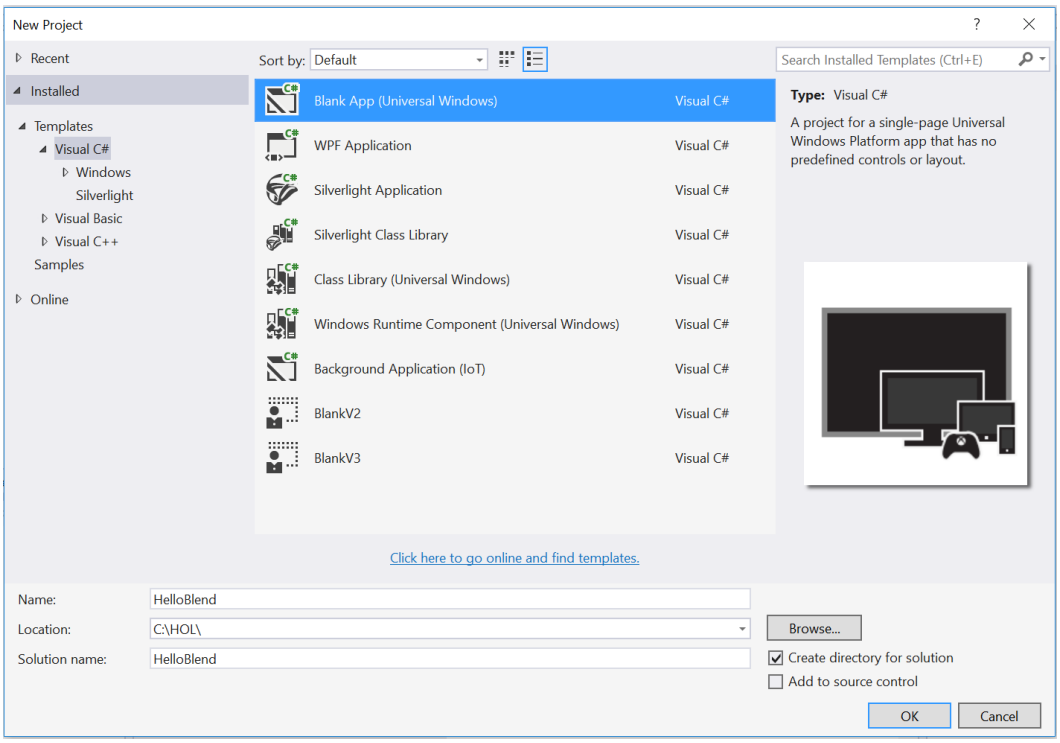

Figure 16

Create the HelloBlend project in Blend.

Note: Although you created this project in Blend, it is a fully functioning Visual Studio project.

4. Blend will display the design for MainPage.xaml in your new project. The Blank App template provides an empty canvas for you to get started.

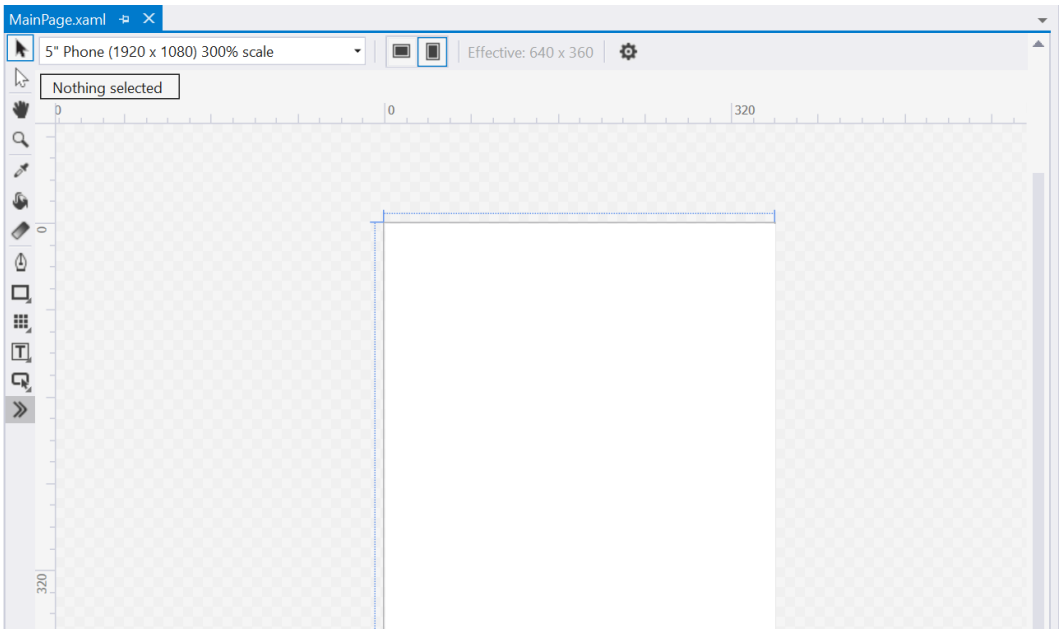

Figure 17

The MainPage of the HelloBlend project shows an empty view.

Task 2 – Generate sample data

Sample data can help to jumpstart your app development. You can quickly generate sample data in Blend through the **Data** panel. In this task, you will generate sample data and view it in a running application.

1. With your **HelloBlend** project open in Blend, go to the **Data** panel. In the default window layout, the Data panel shares a pane with the Solution Explorer.

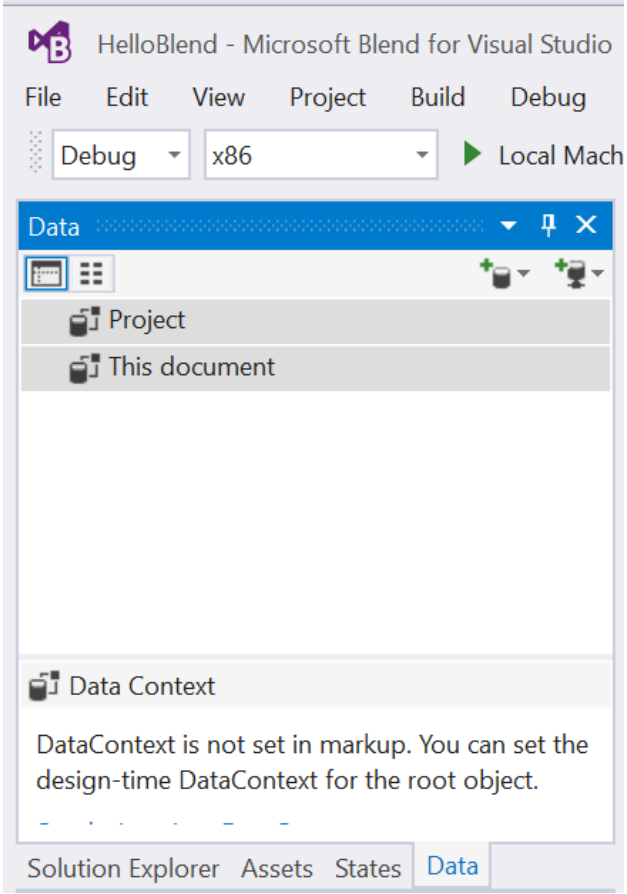

Figure 18

The Data panel in Blend.

Note: If you lose or close a panel, you can type its name into the Quick Launch search to easily bring it back.

2. In the Data Panel, open the **Create sample data** menu and choose **New Sample Data**.

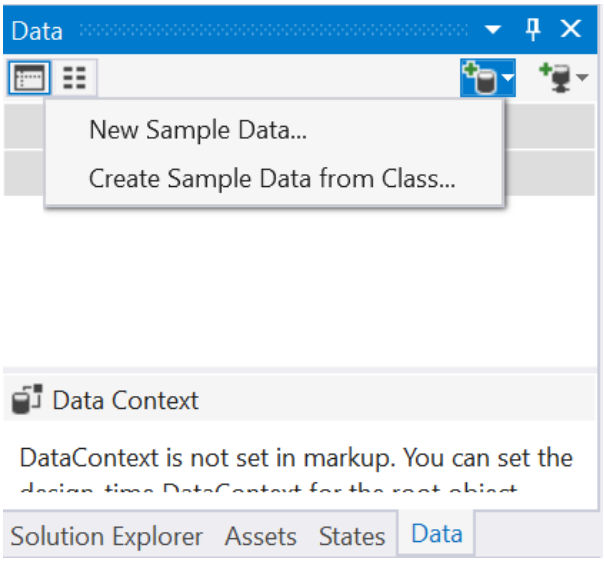

Figure 19

Create sample data from the Data panel.

Note: This tool also allows you to **Create Sample Data from Class** in a project where you have already defined a data schema.

3. When the **New Sample Data** dialog opens, give your data source a name and choose whether to define it for the project or the current document. If you choose **Project**, the sample data will be available to all documents in the project. For this demo, we will only make the sample data available to the current XAML document, which is MainPage.

4. Use the checkbox to enable sample data when the application is running. Click OK to generate the sample data.

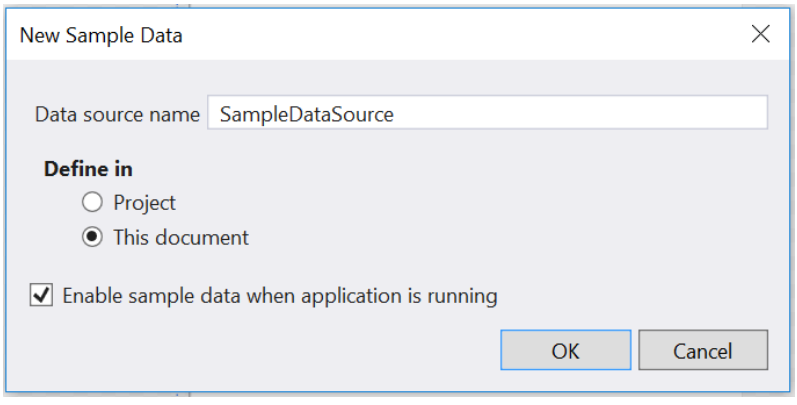

Figure 20

Define the sample data for the current XAML document.

5. Your sample data schema will appear in the **Data** panel under **This document**. There may be a slight delay before the schema is loaded. When it appears, click and drag on the **Items** node to hover your mouse over the MainPage design. A tooltip will appear to preview the controls that will display in your app if you drop the node onto the canvas.

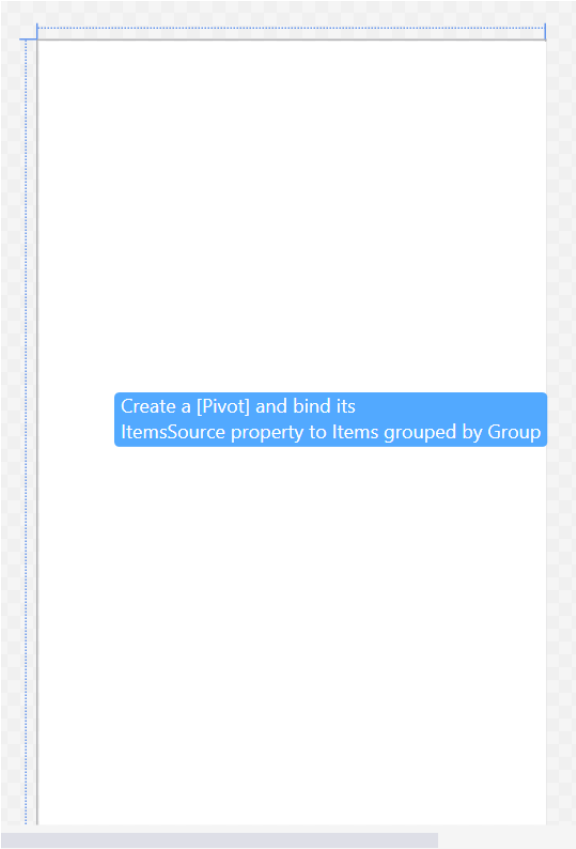

Figure 21

Tooltips help you preview sample data before adding it to your XAML view.

6. Drop the **Items** node onto the **MainPage** canvas. You will see the sample data appear in the designer.

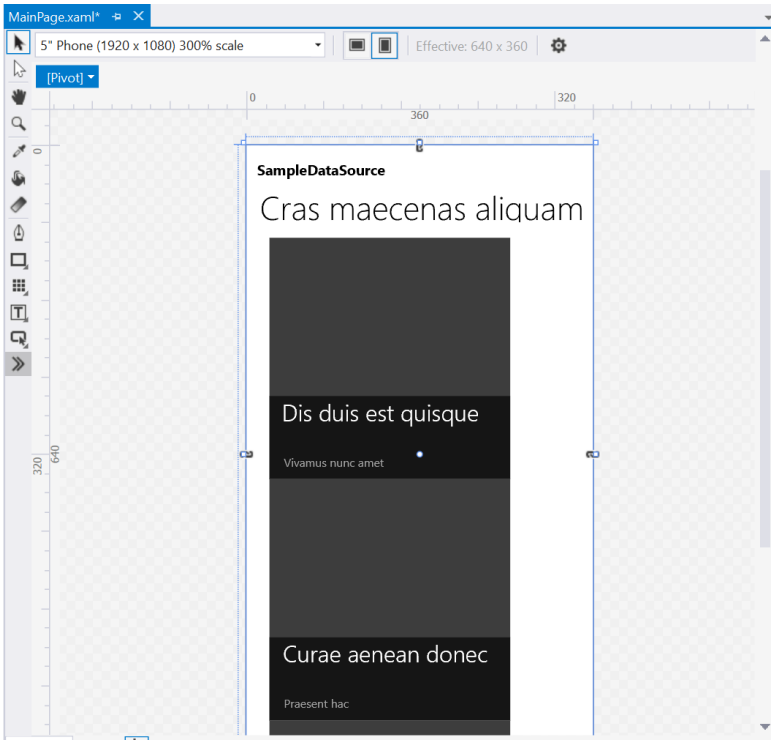

Figure 22

Sample data in the designer.

7. Build and run your app on the **Local Machine**. Use the same debug settings that you would use in Visual Studio. Your sample data will appear live in the app.

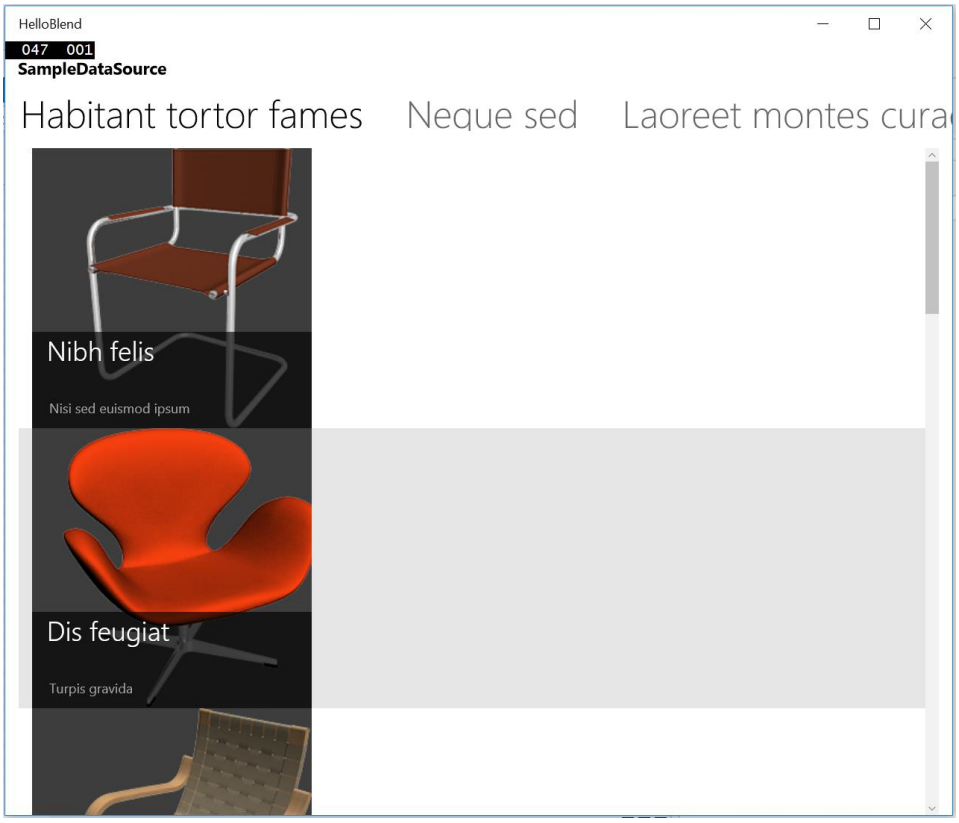

Figure 23
Sample data in the running application.

8. Stop debugging and return to Visual Studio.

9. Optional: Build and run your app in the Mobile Emulator or an IoT device to see how the data displays in different device families.

Summary

The Universal Windows Platform is a powerful collection of core APIs that allows you to target a wide range of devices with a single app. In this lab, you evolved a blank app created from a template into a Hello World app that displays device-specific information across all Windows 10 devices. You also learned how to leverage sample data in Blend to quickly start building and visualizing your app UI. In the next lab, you will learn how to navigate within a UWP app, handle back navigation with the shell-drawn back button, and implement custom back and forward navigation controls.