Windows 10

Hands-on lab

Lab: Live Tiles and Notifications

October 2015

Microsoft

CONTENTS

Overview

Windows 10 provides unique and engaging ways to interact with users outside of the traditional app experience. You can now define the content to display on Live Tiles using a new adaptive markup, allowing you to present the best experience per tile size and screen density. Toast notifications can include interactive and adaptive elements, images, and can synchronize with your Live Tiles.

Objectives

This lab will show you how to:

- Set assets artwork and a background color for your default tiles
- Use BadgeUpdateManager to update the tile badge count
- Create adaptive templates for Live Tiles
- Update Live Tiles
- Support small, medium, wide, and large tiles
- Create an interactive Toast that includes user selectable elements
- Handle the toast action with background activation

System requirements

You must have the following to complete this lab:

- Microsoft Windows 10
- Microsoft Visual Studio 2015

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.

Exercises

This Hands-on lab includes the following exercises:

1. Customize the Default Tile
2. Launch Interactive Toast
3. Schedule Tile Updates

Estimated time to complete this lab: **45 to 60 minutes**.

Exercise 1: Customize the Default Tile

Windows 10 apps use the logo assets you provide to display default tiles for your users. In this exercise, you will import logo assets to create simple tiles and update the tile badge with the BadgeUpdateManager.

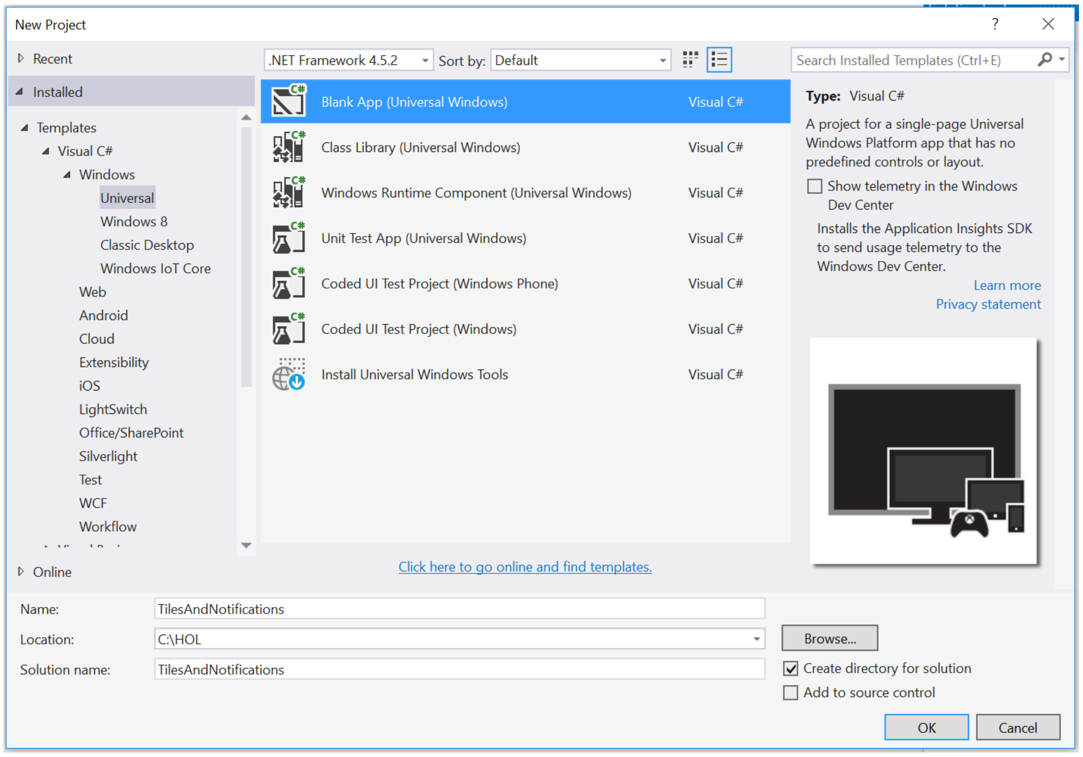
Task 1 – Create a blank Universal Windows app

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.

2. Name your project **TilesAndNotifications** and select the file system 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.

   Leave the options selected to **Create new solution** and **Create directory for solution**. You may deselect both **Add to source control** and **Show telemetry in the Windows Dev Center** if you don't wish to version your work or use Application Insights. Click **OK** to create the project.



.NET Framework 4.5.2 Sort by: Default

Blank App (Universal Windows)	Visual C#
Class Library (Universal Windows)	Visual C#
Windows Runtime Component (Universal Windows)	Visual C#
Unit Test App (Universal Windows)	Visual C#
Coded UI Test Project (Windows Phone)	Visual C#
Coded UI Test Project (Windows)	Visual C#
Install Universal Windows Tools	Visual C#

Type: Visual C#

A project for a single-page Universal Windows Platform app that has no predefined controls or layout.

☐ Show telemetry in the Windows Dev Center

Installs the Application Insights SDK to send usage telemetry to the Windows Dev Center.

Name: TilesAndNotifications

Location: C:\HOL

Solution name: TilesAndNotifications

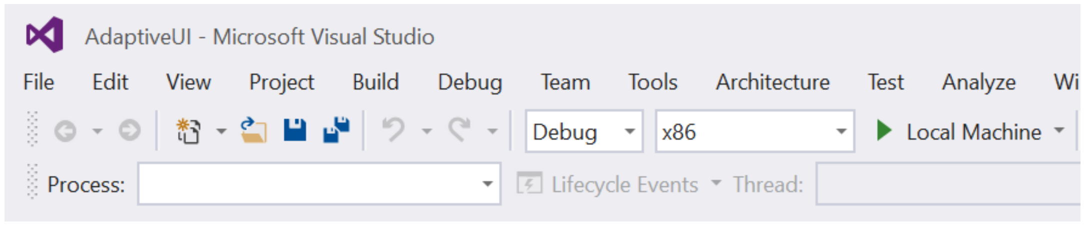
☑ Create directory for solution

☐ Add to source control

Figure 1

Create a new Blank App project in Visual Studio 2015.

3. Set your Solution Configuration to **Debug** and your Solution Platform to **x86**. Select **Local Machine** from the Debug Target dropdown menu.



Debug x86 Local Machine

Process:

Figure 2

Configure your app to run on the Local Machine.

4. Build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.

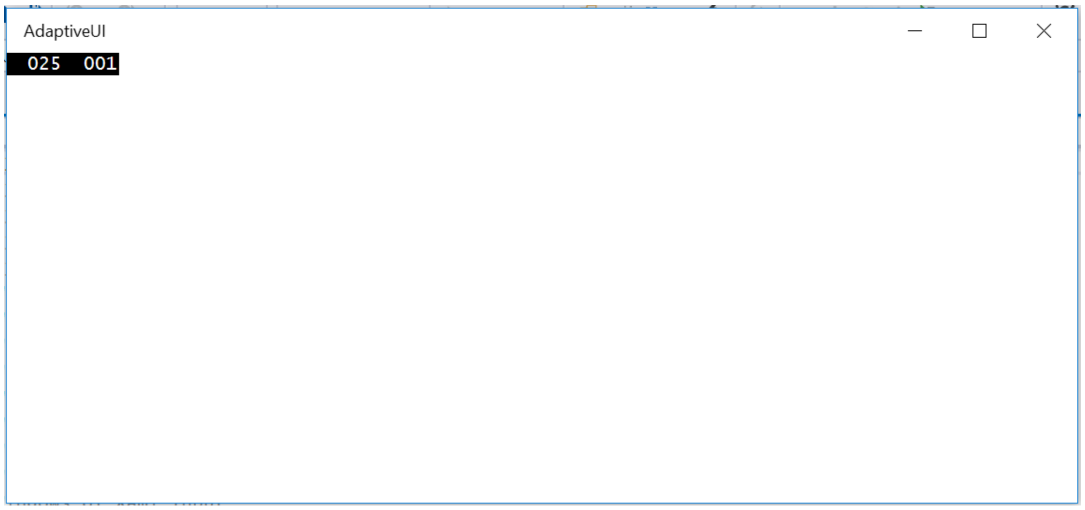


Figure 3

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, you may turn it off by setting **this.DebugSettings.EnableFrameRateCounter** to **false**.

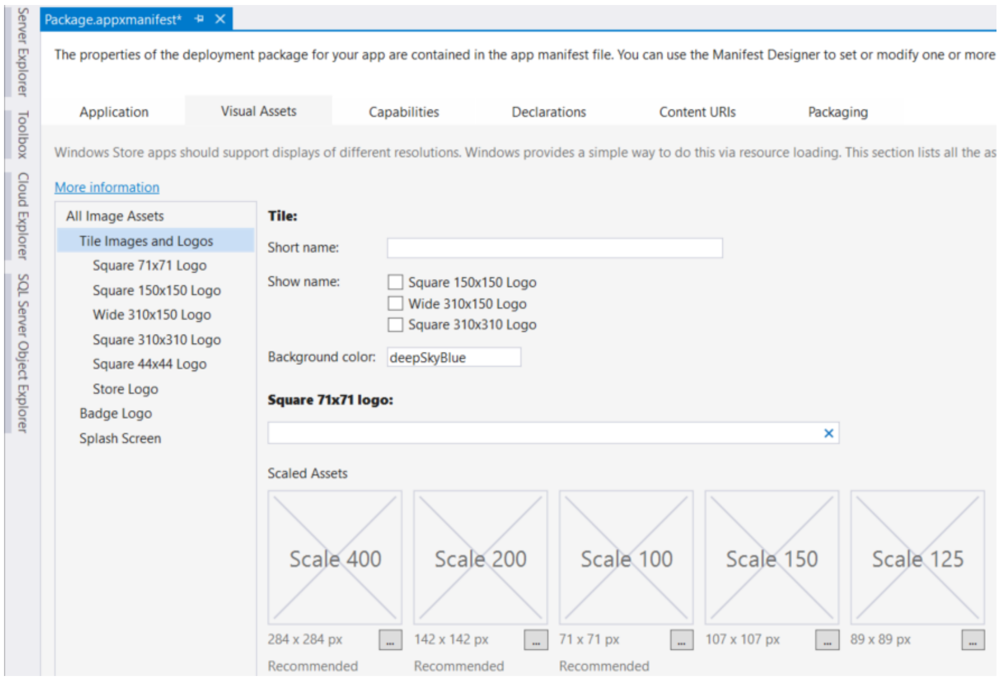
5. Return to Visual Studio and stop debugging.

Task 2 – Import visual assets

When you create a new project, it includes generic tile assets in the Assets folder. To brand your app, you can replace the generic assets from the Blank App template with your own logos. Visual assets may be provided in a number of scale factors for different screen pixel densities. If you provide images at only one scale factor, it is recommended to provide them at scale 200 (200% of the original size for a typical 96dpi device) which should give good results as Windows scales them up or down to accommodate different screen densities on different devices. You should consider creating images at more than one scale factor to give the very best results on different screen densities.

In this task, you will add branded image assets at scale 200 to the app to display on default tiles.

1. Open **Package.appxmanifest** in the manifest editor and navigate to the **Visual Assets** tab. Select **Tile Images and Logos** assets in the image assets pane.



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

Windows Store apps should support displays of different resolutions. Windows provides a simple way to do this via resource loading. This section lists all the as

Tile:

Short name:

Show name: ☐ Square 150x150 Logo ☐ Wide 310x150 Logo ☐ Square 310x310 Logo

Background color: deepSkyBlue

Square 71x71 logo:

Scaled Assets

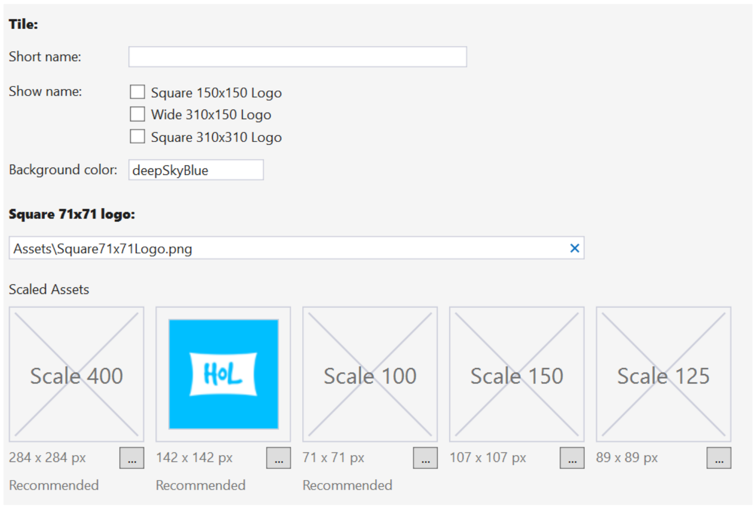
284 x 284 px Recommended | 142 x 142 px Recommended | 71 x 71 px Recommended | 107 x 107 px | 89 x 89 px

Figure 4
Selecting Tile Images and Logos in the Visual Assets tab of the manifest editor.

2. In the Background color field, enter **deepskyblue**.

Note: If you leave the Background color field set to transparent, your tiles will inherit the user's selected accent color, which the user can change in **Windows 10 Settings > Personalization > Colors**. We will set a background color for the tiles in this exercise to make it easier to see the uploaded images files in the manifest editor.

3. Use the ellipsis symbol under the **Square71x71Logo** image at **scale 200** to open the **Select Image** dialog. Navigate to your Hands-on labs **Lab Assets** folder and select the **Square71x71Logo.scale-200.png** image file. Click **Open** to add the image as a visual asset.


Tile:

Short name:

Show name: ☐ Square 150x150 Logo ☐ Wide 310x150 Logo ☐ Square 310x310 Logo

Background color: deepSkyBlue

Square 71x71 logo:

Assets\Square71x71Logo.png

Scaled Assets

Scale 400	HoL	Scale 100	Scale 150	Scale 125
284 x 284 px	142 x 142 px	71 x 71 px	107 x 107 px	89 x 89 px
Recommended	Recommended	Recommended		

Figure 5
The Square71x71Logo asset in the manifest editor.

4. Repeat **Step 2** to add **Square150x150Logo.scale-200.png**, **Wide310x150Logo.scale-200.png, Square310x310Logo.scale-200.png**, and **Square44x44Logo.scale-200.png** to your visual assets. In the case of **StoreLogo.png**, add it to your assets at **scale 100**. In each case, if you are replacing one of the generic tile assets already in the project, Visual Studio will ask you ‘Do you want to replace it?’. Click Yes.

5. Select **Splash Screen** in the image assets pane. In Splash screen configuration, set the Background color field to **deepskyblue**.
   Use the ellipsis symbol under the Scale 200 splash screen image to select **SplashScreen.scale-200.png** from the Lab Assets folder.

6. Build and run your app. Once it has deployed, find your app in the Start menu. Right-click on the app name and choose **Pin to Start**.

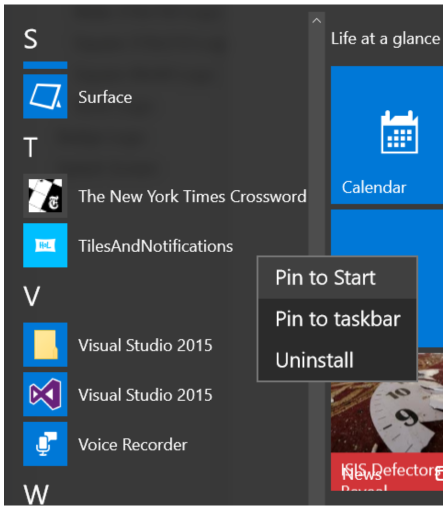


Figure 6
Pin your app to the Start menu.

7. Your default tile will appear on the Start menu with the logo assets and background color you selected in the manifest editor. Right-click on the tile and use the **Resize** option to view your tile at different sizes.

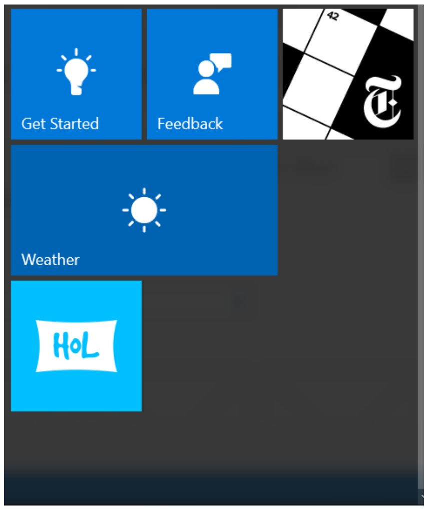


Figure 7
Pin your app to the Start menu.

8. Stop debugging and return to Visual Studio.

Task 3 – Update the tile badge count

Tile badge counts provide a great way to display information at a glance, such as a count of new items. You can update the tile badge count on the default tile, which is a quick and easy way to add value to your app's presence on the start screen.

1. Right-click on your project name and choose **Add > New Folder**. Name the folder **Services**.

2. Right-click on the Services folder and choose **Add > Class**. Name the class **TileService.cs**.

3. Open **TileService.cs** and make it a **public** class.

   C#

   ```
   namespace TilesAndNotifications.Services
   {
       public class TileService
       {
       }
   }
   ```

4. Add a static method to update the badge count on the tile. Add `using` statements for `Windows.Data.Xml.Dom` and `Windows.UI.Notifications`.

   C#

   ```
   using Windows.Data.Xml.Dom;
   using Windows.UI.Notifications;

   namespace TilesAndNotifications.Services
   {
       public class TileService
       {
           static public void SetBadgeCountOnTile(int count)
           {
               // Update the badge on the real tile
               XmlDocument badgeXml =
   BadgeUpdateManager.GetTemplateContent(BadgeTemplateType.BadgeNumber);

               XmlElement badgeElement =
   (XmlElement)badgeXml.SelectSingleNode("/badge");
               badgeElement.SetAttribute("value", count.ToString());

               BadgeNotification badge = new BadgeNotification(badgeXml);

   BadgeUpdateManager.CreateBadgeUpdaterForApplication().Update(badge);
           }
       }
   }
   ```

5. Return to MainPage.xaml. Create a button to update the badge count.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Button Click="UpdateBadge" VerticalAlignment="Top" Margin="12">Update
Badge Count</Button>
</Grid>
```

6. In the MainPage code behind, add the **_count** field of type `int` and the **UpdateBadge()** method to call the TileService. Add the **TilesAndNotifications.Services** namespace.

C#

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

int _count;

private void UpdateBadge (object sender, RoutedEventArgs e)
{
    _count++;
    TileService.SetBadgeCountOnTile(_count);
}
```

7. Build and run your app on the Local Machine. Once it has deployed, find your app in the Start menu again.

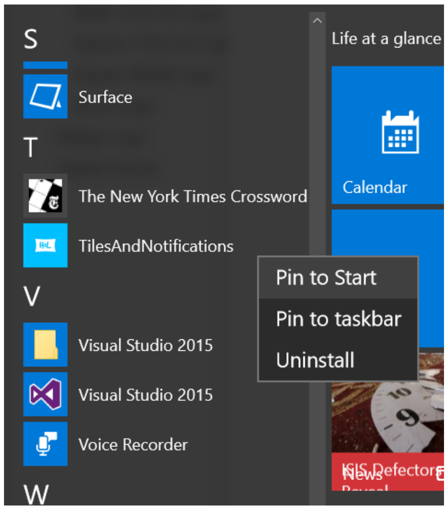


Figure 8
Pin your app to the Start menu.

8. In your running app, click the **Update Badge Count** button. When you return to your Live Tile on the Start screen, you will see a badge count of 1. This badge count will increment every time you call the **UpdateBadge()** method.

9. Stop debugging and return to Visual Studio.

Exercise 2: Create Adaptive Live Tiles

Live Tiles in Windows 10 use adaptive templates to deliver content customized to a device and screen density. While legacy templates are still compatible with live tiles, adaptive templates give you more freedom to choose how your content will display on all devices. Groups and subgroups allow you to semantically link content within the tile. In this exercise, you will create an adaptive layout and display it with static data on the tile.

Task 1 – Add a model

In a typical app, a Live Tile would display existing app data. In this task, we will create a class to mock up static data to display on your tiles.

1. Right-click on the project name and create a folder called **Models**.

2. Add a class named **PrimaryTile.cs** to the Models folder.

3. Open the **PrimaryTile** class and make it public. Add static data as string fields to define data that we will use to populate the tile.

C#

```
namespace TilesAndNotifications.Models
{
    public class PrimaryTile
    {
        public string time { get; set; } = "8:15 AM, Saturday";
        public string message { get; set; } = "Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.";
        public string message2 { get; set; } = " At vero eos et accusamus et
iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti
atque corrupti quos dolores et quas molestias excepturi sint occaecati
cupiditate non provident.";
        public string branding { get; set; } = "name";
        public string appName { get; set; } = "HoL";
    }
}
```

Task 2 – Build the tile XML

The adaptive tile schema is written in XML. In this task, you will generate the XML necessary to display text content from the PrimaryTile model on both small and medium Live Tiles.

1. In **TileService.cs**, add the **TilesAndNotifications.Models** and **System.Xml.Linq** namespaces.

C#

```
using TilesAndNotifications.Models;
using System.Xml.Linq;
```

2. Add a **CreateTiles** method to generate the XML for the small and medium tiles.

C#

```
public static Windows.Data.Xml.Dom.XmlDocument CreateTiles (PrimaryTile
primaryTile)
{
    XDocument xDoc = new XDocument(
        new XElement("tile", new XAttribute("version", 3),
            new XElement("visual",
                // Small Tile
                new XElement("binding", new XAttribute("branding",
primaryTile.branding), new XAttribute("displayName", primaryTile.appName), new
XAttribute("template", "TileSmall"),
                    new XElement("group",
                        new XElement("subgroup",
                            new XElement("text", primaryTile.time, new
XAttribute("hint-style", "caption")),
                            new XElement("text", primaryTile.message, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap", true),
new XAttribute("hint-maxLines", 3))
                        )
                    )
                ),

                // Medium Tile
                new XElement("binding", new XAttribute("branding",
primaryTile.branding), new XAttribute("displayName", primaryTile.appName), new
XAttribute("template", "TileMedium"),
                    new XElement("group",
                        new XElement("subgroup",
                            new XElement("text", primaryTile.time, new
XAttribute("hint-style", "caption")),
                            new XElement("text", primaryTile.message, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap", true),
new XAttribute("hint-maxLines", 3))
                        )
                    )
                )
            )
        )
    );

    Windows.Data.Xml.Dom.XmlDocument xmlDoc = new
Windows.Data.Xml.Dom.XmlDocument();
    xmlDoc.LoadXml(xDoc.ToString());
return xmlDoc;
}
```

Note: There are a number of elements you can include in your adaptive tile schema. You may choose among preset styles for each element type. For more information, visit https://msdn.microsoft.com/en-us/library/windows/apps/Mt186446.aspx.

For instructions and examples, see this blog post: Adaptive tiles schema and documentation.

3. Open MainPage.xaml. Add a button below the Update Badge Count button to update the primary tile. We will enclose the buttons in a StackPanel to facilitate layout.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <Button Click="UpdateBadge" VerticalAlignment="Top" Margin="12">Update
Badge Count</Button>
        <Button Click="UpdatePrimaryTile" VerticalAlignment="Top"
Margin="12">Update Primary Tile</Button>
    </StackPanel>
</Grid>
```

4. In the MainPage code-behind, add the **TilesAndNotifications.Models** and **Windows.UI.Notificiations** namespaces.

C#

```
using TilesAndNotifications.Models;
using Windows.UI.Notifications;
```

5. Add the **UpdatePrimaryTile()** method to the MainPage code-behind.

C#

```
private void UpdatePrimaryTile(object sender, Windows.UI.Xaml.RoutedEventArgs
e)
{
    var xmlDoc = TileService.CreateTiles(new PrimaryTile());

    var updater = TileUpdateManager.CreateTileUpdaterForApplication();
    TileNotification notification = new TileNotification(xmlDoc);
    updater.Update(notification);
}
```

Note: Every time you update a tile, you are actually creating a new instance of that tile. When creating the tile in an app with live data, you can pass in the latest data that you want to display on the tile.

6. Build and run your app. Pin the primary tile for your app to the Start menu if it is not already present.

7. Use the **Update Primary Tile** button in your running app to trigger the tile update. Open the Start menu on your device and wait until you see the default tile update as the Live Tile flips into view. Resize the tile to see the effect for both the Small and the Medium tile sizes.

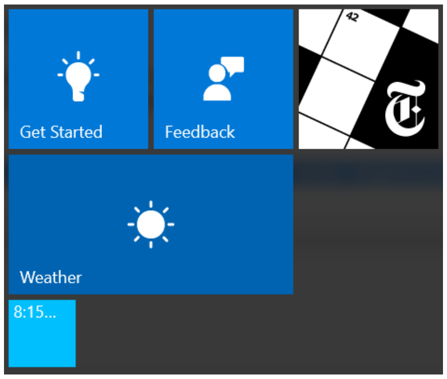


Figure 9
The small Live Tile.

8. Resize your primary tile to **Medium**. This time, when the Live Tile flips in, you will see more information provided by the adaptive template.

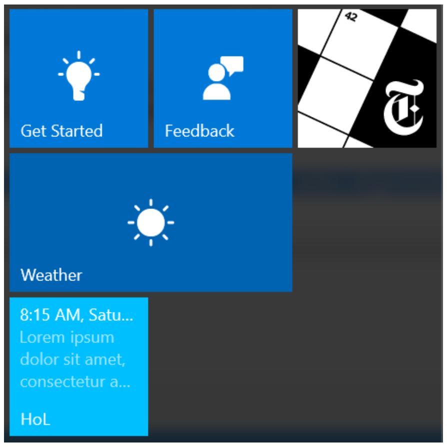


Figure 10
The medium Live Tile.

9. Stop debugging and return to Visual Studio.

Task 3 – Create adaptive templates for Wide and Large tiles.

1. Add the Wide and Large tiles to the **CreateTiles()** method. Larger live tiles have room to display images in addition to text. In this case, we will display an image that already exists in your Assets folder. Make sure to add a comma after the Medium tile XML to continue the list.

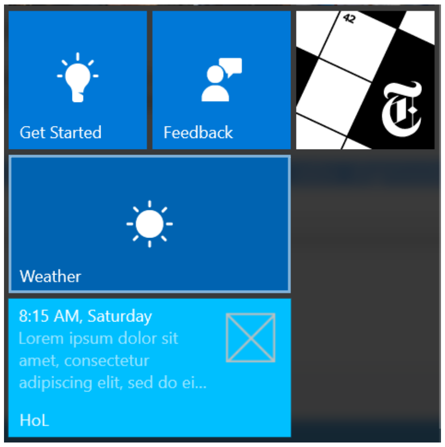
C#

```
),

// Wide Tile
new XElement("binding", new XAttribute("branding", primaryTile.branding), new
XAttribute("displayName", primaryTile.appName), new XAttribute("template",
"TileWide"),
    new XElement("group",
        new XElement("subgroup",
            new XElement("text", primaryTile.time, new XAttribute("hint-
style", "caption")),
            new XElement("text", primaryTile.message, new XAttribute("hint-
style", "captionsubtle"), new XAttribute("hint-wrap", true), new
XAttribute("hint-maxLines", 3)),
            new XElement("text", primaryTile.message2, new XAttribute("hint-
style", "captionsubtle"), new XAttribute("hint-wrap", true), new
XAttribute("hint-maxLines", 3))
            ),
        new XElement("subgroup", new XAttribute("hint-weight", 15),
            new XElement("image", new XAttribute("placement", "inline"), new
XAttribute("src", "Assets/StoreLogo.png"))
        )
    )
),

//Large Tile
new XElement("binding", new XAttribute("branding", primaryTile.branding), new
XAttribute("displayName", primaryTile.appName), new XAttribute("template",
"TileLarge"),
    new XElement("group",
        new XElement("subgroup",
            new XElement("text", primaryTile.time, new XAttribute("hint-
style", "caption")),
            new XElement("text", primaryTile.message, new XAttribute("hint-
style", "captionsubtle"), new XAttribute("hint-wrap", true), new
XAttribute("hint-maxLines", 3)),
            new XElement("text", primaryTile.message2, new XAttribute("hint-
style", "captionsubtle"), new XAttribute("hint-wrap", true), new
XAttribute("hint-maxLines", 3))
            ),
        new XElement("subgroup", new XAttribute("hint-weight", 15),
            new XElement("image", new XAttribute("placement", "inline"), new
XAttribute("src", "Assets/StoreLogo.png"))
```

```
          )
      )
)
```

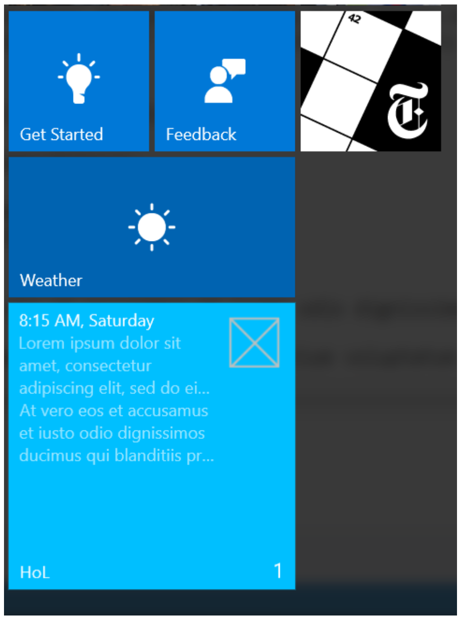
Note: To display wide and large Live Tiles, you must have a WideLogo and Square310x310Logo assets defined in your app manifest. We added these assets in Exercise 1.

2. Build and run your app. Click the Update Primary Tile button. Pin the primary tile and resize it to **Wide** and **Large** sizes. Notice that while both have the ability to display **message** and **message2**, the large tile is more likely to have room to display it.



8:15 AM, Saturday
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do ei...

Figure 11
The wide Live Tile.



8:15 AM, Saturday
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do ei...
At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis pr...

Figure 12

The large Live Tile has room to display the most content.

Note: Large tiles are not available on devices running Windows 10 Mobile.

3. Stop debugging and return to Visual Studio.

Exercise 3: Interactive Toast

In addition to adaptive tiles, interactive and adaptive toast are new in Windows 10. Toast notifications can include content, inline images, and actions for user input. Foreground actions open the app to complete a task, while background actions are handled without needing to launch the app.

In this exercise, you will create a toast with action buttons that will update data in your app via a background task. The app will contain a mock to-do item consisting of a checkbox and description. The toast will allow you to mark the item as completed or not completed, which will become visible when you refresh the app main page.

Task 1 – Create the toast service

Toast XML is built similarly to the Tile XML you created in the previous exercise. We will create a service class to create the XML for the toast notifications and call it from the MainPage code-behind.

1. Right-click on the **Services** folder and choose **Add > Class**. Name the class **ToastService.cs**.

2. Make the class both **public** and **static**, and add the namespaces **System.Xml.Linq** and **Windows.Data.Xml.Dom**.

C#

```
using System.Xml.Linq;
using Windows.Data.Xml.Dom;

namespace TilesAndNotifications.Services
{
    public static class ToastService
```

3. Add the **CreateToast()** function to build and load the XmlDocument.

C#

```
public static class ToastService
{
    public static XmlDocument CreateToast()
    {
        var xDoc = new XDocument(
            new XElement("toast",
                new XElement("visual",
                    new XElement("binding", new XAttribute("template",
"ToastGeneric"),
                        new XElement("text", "To Do List"),
                        new XElement("text", "Is the task complete?")
                        ) // binding
                    ), // visual
                new XElement("actions",
                    new XElement("action", new XAttribute("activationType",
"background"),
                        new XAttribute("content", "Yes"), new
XAttribute("arguments", "yes")),
                    new XElement("action", new XAttribute("activationType",
"background"),
                        new XAttribute("content", "No"), new
XAttribute("arguments", "no"))
                    ) // actions
                )
            );

        var xmlDoc = new XmlDocument();
        xmlDoc.LoadXml(xDoc.ToString());
        return xmlDoc;
    }
}
```

This code creates the following toast definition using the new adaptive XML schema:

```
<toast>
  <visual>
    <binding template="ToastGeneric">
      <text>To Do List</text>
      <text>Is the task complete?</text>
    </binding>
  </visual>
  <actions>
    <action activationType="background" content="Yes" arguments="yes" />
    <action activationType="background" content="No" arguments="no" />
  </actions>
</toast>
```

Note: The **action** buttons aren't wired up yet, but we will need them later in the exercise. The **activationType** of **background** means the actions will not launch the app in the foreground when activated. For more on the Toast XML schema, visit https://msdn.microsoft.com/en-us/library/windows/apps/br230849.aspx

4. Save and close the ToastService.

5. Open **MainPage.xaml**. Add row definitions to the Grid with a second StackPanel in the bottom row.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <StackPanel Grid.Row="0" Margin="12">
        <TextBlock Text="Adaptive Tiles" FontSize="20" FontWeight="Light" />
        <Button Click="UpdateBadge" VerticalAlignment="Top" Margin="12">Update 
Badge Count</Button>
        <Button Click="UpdatePrimaryTile" VerticalAlignment="Top" 
Margin="12">Update Primary Tile</Button>
    </StackPanel>
    <StackPanel Grid.Row="1" Margin="12">

    </StackPanel>
</Grid>
```

6. Add a section title and Notify button to the new StackPanel. You will create the Notify() method in the next step.

XAML

```
<StackPanel Grid.Row="1" Margin="12">
    <TextBlock Text="Interactive Toast" FontSize="20" FontWeight="Light" />
    <Button Click="Notify" Margin="12">Notify</Button>
</StackPanel>
```

7. Add the **Windows.UI.Notifications** and **TilesAndNotifications.Services** namespaces to the MainPage code-behind.

C#

```
using TilesAndNotifications.Services;
```

8. Create the **Notify()** method in the code-behind.

C#

```
private void Notify(object sender, RoutedEventArgs e)
{
    var xmlDoc = ToastService.CreateToast();
    var notifier = ToastNotificationManager.CreateToastNotifier();
    var toast = new ToastNotification(xmlDoc);
    notifier.Show(toast);
}
```

9. Build and run your app on the local machine. Use the **Notify** button to send a toast. When the toast arrives, you can click on the action buttons, but nothing will happen yet. We will wire up the actions in a later task.

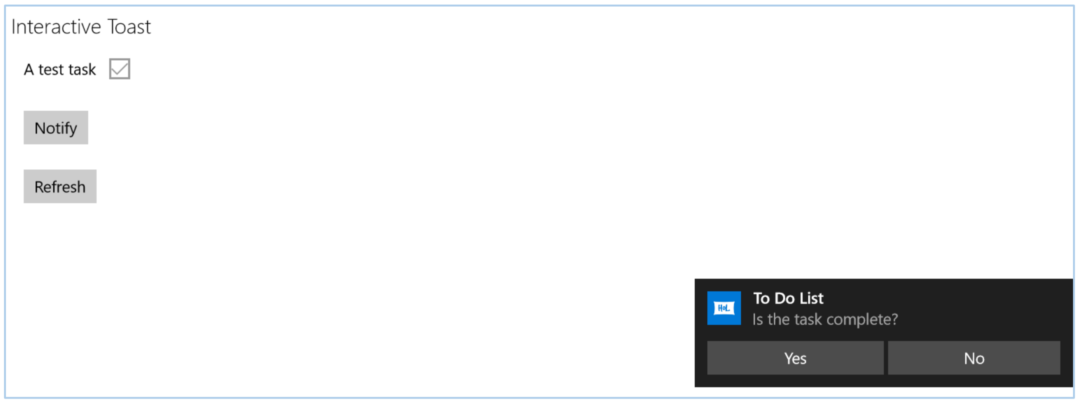


Figure 13
The toast with action buttons.

10. Stop debugging and return to Visual Studio.

Task 2 – Create the model and helper

Before we can implement the toast actions, we need a simple class and helper for our mock to-do item. We will create these in a class library to make them accessible from any project in the solution. Both the app and the background task, which will exist within a Windows Runtime component, will need access to the model.

1. Right-click on the Solution in the Solution Explorer and Add > New Project. Choose a project type of **Visual C# > Windows > Universal > Class Library (Universal Windows)**. Give it the name **TasksAndNotifications.Library**.

2. Delete **Class1.cs** using the Solution Explorer context menu. If prompted, confirm that you would like to delete it permanently.

3. Right-click on the **TilesAndNotifications.Library** project and choose **Add > Existing Item**. Browse to the **Lab Assets** folder in your Hands-on Labs directory and select **ToDoTask.cs** and **ToDoTaskFileHelper.cs**. Add them to your class library.

4. Open **ToDoTask.cs**.You will see that a simple ToDo item consists of an **Id**, **Description**, and **IsComplete** flag. There are two methods in the class to handle serialization to and from JSON.

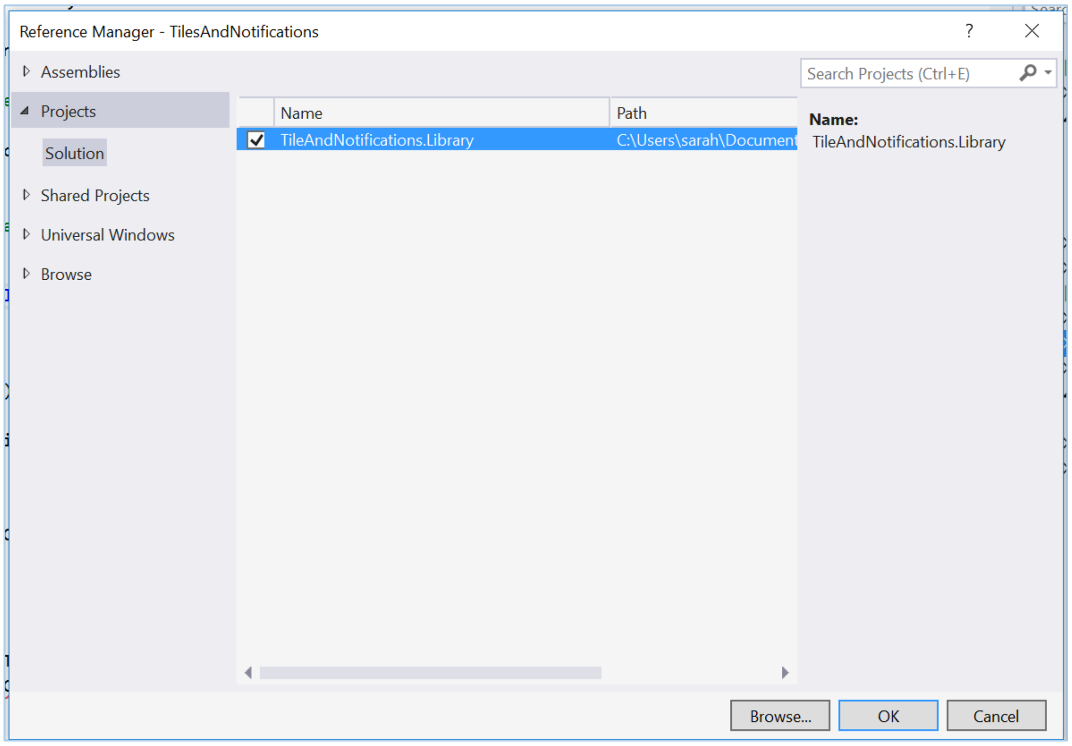
Note: We will store the ToDo task data in a JSON file to make it accessible to the background task, which can access files in the app package. You will create the background task later in this exercise.

5. Open **ToDoTaskFileHelper.cs**. This helper saves the serialized JSON to a file and reads it back in from the file when needed.

6. Return to your app in Solution Explorer. Right-click on the **References** folder and add **TilesAndNotifications.Library** as a reference.

7. Right-click on the **Assets** folder and choose **Add > Existing Item**. Browse the Lab Assets folder and add the **task.json** starter file.

8. Right-click on the **task.json** file and open its **Properties**. Set its **Build Action** to **Content** and set **Copy to Output Directory** to **Copy Always**. Close the Properties pane.



Name	Path
☑ TileAndNotifications.Library	C:\Users\sarah\Document

Name:
TileAndNotifications.Library

Figure 14
Add the class library as a reference.

Task 3 – Display the ToDo item and implement INotifyPropertyChanged

We will display the ToDo item on the MainPage with a checkbox to indicate the IsComplete status. The ToDo task will be bound to a property in the code behind. To make it possible for our UI to be notified when our background task makes a change to the ToDo task later in this exercise, we will implement **INotifyPropertyChanged** on the CurrentToDoTask.

1. Open **MainPage.xaml.cs**. Add the **TilesAndNotifications.Library** and the **System.ComponentModel** namespaces.

C#

```
using TilesAndNotifications.Library;
using System.ComponentModel;
```

2. Add a private field for a ToDoTask and a public property to access it.

C#

```
private int _count;
private ToDoTask _currentToDoTask;

public MainPage()
{
    InitializeComponent();
    Loaded += MainPage_Loaded;
}

public ToDoTask CurrentToDoTask
{
    get { return _currentToDoTask; }
    set
    {
        _currentToDoTask = value;
    }
}
```

3. Implement **INotifyPropertyChanged** to notify the UI when the **CurrentToDoTask** changes. We will bind the UI in a later step.

C#

```
public sealed partial class MainPage : Page, INotifyPropertyChanged
{
…
    public ToDoTask CurrentToDoTask
    {
        get { return _currentToDoTask; }
        set
        {
            _currentToDoTask = value;
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(CurrentToDoTask)));
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

Note: INotifyPropertyChanged provides generic property-change notifications to clients, which are typically bound to the value that is changing. In this demo, our UI will need to know when the background task triggered by the toast updates the ToDo item. For more on INotifyPropertyChanged, visit https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.data.inotifypropertychanged

4. Open MainPage.xaml and add a TextBlock and CheckBox to display the ToDo task and its status. Add a button to refresh the data. You will wire up the button in the next step.

XAML

```
<StackPanel Grid.Row="1" Margin="12">
    <TextBlock Text="Interactive Toast" FontSize="20" FontWeight="Light" />
    <StackPanel Orientation="Horizontal" Margin="12">
        <TextBlock x:Name="Description" VerticalAlignment="Center"
Text="{x:Bind CurrentToDoTask.Description, Mode=OneWay}"/>
        <CheckBox Margin="12,0,0,0" IsChecked="{x:Bind
CurrentToDoTask.IsComplete, Mode=OneWay}" IsEnabled="False" />
    </StackPanel>
    <Button Click="Notify" Margin="12">Notify</Button>
    <Button Click="{x:Bind Refresh}" Margin="12">Refresh</Button>
</StackPanel>
```

5. Return to the MainPage code-behind. Add the async **Refresh()** method to read the latest task data in from the JSON file.

C#

```
private async void Refresh()
{
    var json = await ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
    CurrentToDoTask = ToDoTask.FromJson(json);
}
```

6. Subscribe to the Loaded event and call the Refresh() method when the page loads.

C#

```
public MainPage()
{
    InitializeComponent();
    Loaded += MainPage_Loaded;
}

…

private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    Refresh();
}

private async void Refresh()
```

```
{
    var json = await ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
    CurrentToDoTask = ToDoTask.FromJson(json);
}
```

7. Build and run your app. You will see the test task show up with the IsComplete checkbox selected by default.

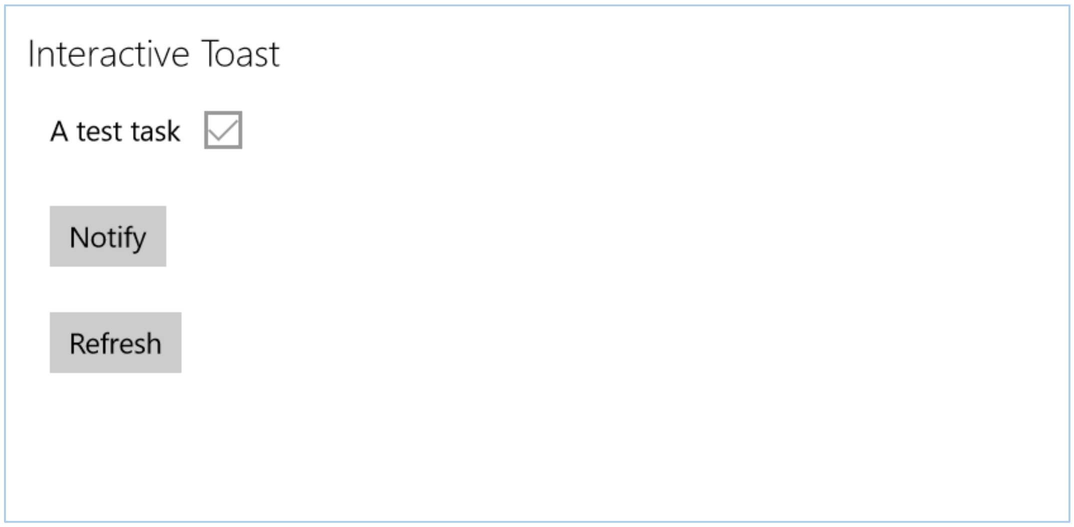


Figure 15
The ToDo task item.

8. Stop debugging and return to Visual Studio.

Task 4 – Create the background task

In this task, you will create the background task as part of Windows Runtime Component.

1. Right-click on the solution in the Solution Explorer and choose Add > New Project. Add a project of type **Visual C# > Windows > Universal > Windows Runtime Component (Universal Windows)**. Give it the name **BackgroundTasks**.

2. Right-click on the **References** folder in the BackgroundTasks project and choose **Add > Reference**. Add the **TilesAndNotifications.Library** project as a reference. The library will give you access to the ToDoTask and its helper.

3. Right-click and **Rename** Class1.cs to **ToastUpdateTask.cs**. When prompted, agree to perform a rename of all references to **Class1** to **ToastUpdateTask**.

4. Open **ToastUpdateTask.cs** and add the **Windows.ApplicationModel.Background**, **Windows.UI.Notifications** and **TilesAndNotifications.Library** namespaces.

C#

```
using Windows.ApplicationModel.Background;
using Windows.UI.Notifications;
using TilesAndNotifications.Library;
```

5. Implement the **IBackgroundTask** interface. Create and complete a deferral within the Run method.

C#

```
namespace BackgroundTasks
{
    public sealed class ToastUpdateTask : IBackgroundTask
    {
        public async void Run(IBackgroundTaskInstance taskInstance)
        {
            var deferral = taskInstance.GetDeferral();

            deferral.Complete();
        }
    }
}
```

6. Convert the **taskInstance.TriggerDetails** to a **ToastNotificationActionTriggerDetails**. These details are triggered by the **Yes** action on the toast. If the details aren't null, read in the JSON task and set its **IsComplete** flag to true.

C#

```
namespace BackgroundTasks
{
    public sealed class ToastUpdateTask : IBackgroundTask
    {
        public async void Run(IBackgroundTaskInstance taskInstance)
        {
            var deferral = taskInstance.GetDeferral();

            var details = taskInstance.TriggerDetails as
ToastNotificationActionTriggerDetail;
            if (details != null)
            {
                string arguments = details.Argument;
                // this is where you would retrieve any user input
                var userInput = details.UserInput;

                var json = await ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
                var task = ToDoTask.FromJson(json);

                task.IsComplete = arguments == "yes";
```

```
                    await ToDoTaskFileHelper.SaveToDoTaskJson(task.ToJson());
                }

                deferral.Complete();
            }
        }
}
```

Note: For more on notification args and trigger details, visit https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.notifications

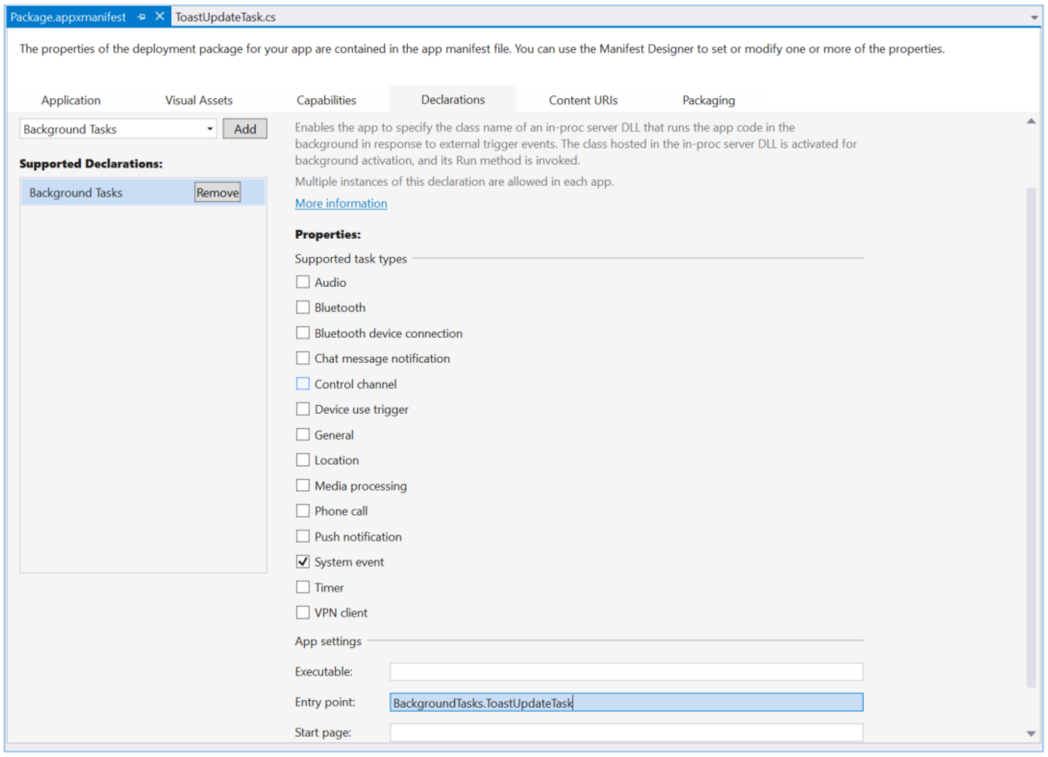
---

Task 5 – Register the background task

Before running the app, we must register the background task in our TilesAndNotifications app and declare it in the manifest.

1. Return to your app project. Open the package manifest in the manifest editor and browse to the **Declarations** tab. Choose **BackgroundTasks** from the list of **Available Declarations** and **Add** the declaration.

2. Set the **task type** to **System Event** and the **Entry point to BackgroundTasks.ToastUpdateTask**. Save and close the manifest.



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

Background Tasks

Supported Declarations:

Background Tasks

Enables the app to specify the class name of an in-proc server DLL that runs the app code in the background in response to external trigger events. The class hosted in the in-proc server DLL is activated for background activation, and its Run method is invoked.

Multiple instances of this declaration are allowed in each app.

Properties:

Supported task types

- [ ] Audio
- [ ] Bluetooth
- [ ] Bluetooth device connection
- [ ] Chat message notification
- [ ] Control channel
- [ ] Device use trigger
- [ ] General
- [ ] Location
- [ ] Media processing
- [ ] Phone call
- [ ] Push notification
- [x] System event
- [ ] Timer
- [ ] VPN client

App settings

Executable:

Entry point: BackgroundTasks.ToastUpdateTask

Start page:

Figure 16

Declare the background task in the app manifest.

3. In your **TilesAndNotifications** project, right-click on the **References** folder and choose **Add > Reference**. Add the **BackgroundTasks** project as a reference.

4. Open **App.xaml.cs**. Add the **Windows.ApplicationModel.Background** and the **BackgroundTasks** namespaces.

C#

```
using Windows.ApplicationModel.Background;
using BackgroundTasks;
```

5. Add an **async** method called **RegisterBgTask** to handle task registration.

C#

```
private async void RegisterBgTask(string taskName, Type taskType, bool
isInternetRequired = false)
        {
            var backgroundAccessStatus = await
BackgroundExecutionManager.RequestAccessAsync();

            if (backgroundAccessStatus ==
BackgroundAccessStatus.AllowedMayUseActiveRealTimeConnectivity ||
                backgroundAccessStatus ==
BackgroundAccessStatus.AllowedWithAlwaysOnRealTimeConnectivity)
            {
                // Check to see if the task has already been registered
                if (BackgroundTaskRegistration.AllTasks.Any(t => t.Value.Name
== taskName))
                {
                    return;
                }

                // Register the toast update task
                var taskBuilder = new BackgroundTaskBuilder
                {
                    Name = taskName,
                    TaskEntryPoint = taskType.FullName
                };

                taskBuilder.SetTrigger(new ToastNotificationActionTrigger());
                var registration = taskBuilder.Register();
            }
        }
```

6. Call **RegisterBgTask** from the App constructor and pass in the typeof **ToastUpdateTask**.

C#

```
public App()
{
    this.InitializeComponent();
    this.Suspending += OnSuspending;
    RegisterBgTask("ToastUpdateTask", typeof(ToastUpdateTask));
}
```

7. Build and run your app. Use the Notify button to trigger a toast. If your task was set to complete, choose **No** as the toast action. Use the Refresh button to refresh the task status. The checkbox will reflect the choice you made in the toast.

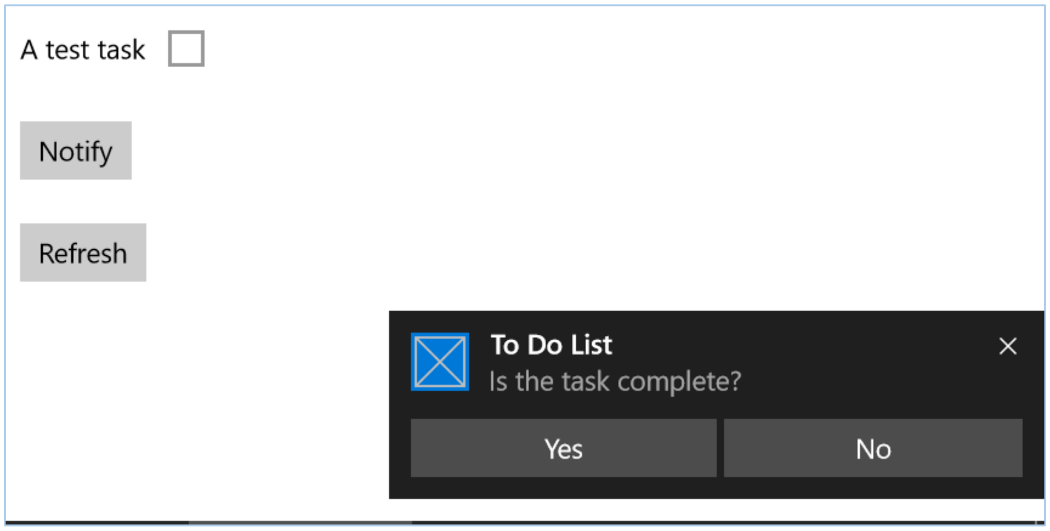


Figure 17
Change the task status from the interactive toast.

8. Stop debugging and return to Visual Studio.

Summary

In this lab, you branded your app with custom visual assets and used them to create a rich experience with default tiles that can display badge counts. You learned about the new adaptive tile schema and created Live Tiles in small, medium, wide, and large sizes, then explored interactive toast with background activation.