

Hands-on lab

Lab 7: Tiles and notifications

September 2012



Contents

[Overview 3](#_Toc334680833)

[Exercise 1: Incorporate secondary tiles 5](#_Toc334680834)

[Task 1 – Modify the app bar 5](#_Toc334680835)

[Task 2 – Modify OnLaunched 7](#_Toc334680836)

[Task 3 – Pin a recipe 7](#_Toc334680837)

[Exercise 2: Incorporate push notifications 8](#_Toc334680838)

[Task 1 – Modify the app manifest 9](#_Toc334680839)

[Task 2 – Subscribe to push notifications 10](#_Toc334680840)

[Task 3 – Test push notifications 11](#_Toc334680841)

[Exercise 3: Incorporate scheduled toasts 12](#_Toc334680842)

[Task 1 – Modify the app bar 12](#_Toc334680843)

[Task 2 – Enable toast notifications 14](#_Toc334680844)

[Task 3 – Make a toast! 15](#_Toc334680845)

[Summary 16](#_Toc334680846)

Overview

* 1. Tiles are an important element of a Windows Store app’s user experience. When an app is installed, its tile is created on the Windows 8 start screen. Known as a *primary tile*, this tile serves as a shortcut for launching the app. By default, the image for a primary tile comes from the Logo.png file included by Visual Studio in a Windows Store app project. That image measures 150 by 150 pixels and creates a *square tile*. However, you can also include a *wide tile* in your app by adding a 310-by-150 image to your project and designating it as the “wide logo” in the app manifest. If an app supports wide tiles, users can toggle between square tiles and wide tiles on the start screen.
  2. Windows Store apps also have the option of creating additional tiles known as *secondary tiles*. Secondary tiles serve as shortcuts to launch an app, and provide it with a predefined location or state. For example, a weather app might allow users to create secondary tiles for geographic locations such as Redmond, WA, and Atlanta, GA. A user may then start the app and view the weather in Redmond or Atlanta simply by tapping the secondary tile.
  3. You can create secondary tiles by using the Windows Runtime’s **Windows.UI.StartScreen.SecondaryTile** class. Tile creation is usually initiated by an action performed by the user, such as tapping a command in the app bar. An app that creates a secondary tile provides a number of parameters, including the URIs of the tile’s background images (both square and wide if you want the user to be able to toggle between the two), and a string that contains activation arguments. When the app is launched from the secondary tile, the operating system passes the activation arguments to the app. These arguments provide the information the app needs to initialize itself. For a weather app, the activation arguments might be nothing more than a zip code.
  4. After a tile is created, it does not have to remain static. Windows Store apps enjoy a number of ways to make a tile’s content both dynamic and compelling via tile updates and push notifications. For example, an app can use the **Windows.UI.Notifications.TileUpdater** class to update the contents of a tile. These updates can optionally be queued. When queuing is enabled, the operating system rotates between the last five updates every few seconds, making the tile feel fresh and alive.
  5. Of course, an app has to be running to use the **TileUpdater** class to update a tile. But tiles can be updated even when the app isn’t running, through the Windows Notification Service, or WNS. Suppose a weather app wants to display severe-weather alerts, even if it is not running. Through WNS, the servers for the weather app can send push notifications that update tiles on the start screen.
  6. Push notifications aren’t limited to updating tiles. They can also pop up a toast window that contains a message for the user (for example, "Severe weather detected in your area”), and they can display a badge – a number or predefined glyph – on a tile. For example, think of a mail app that alerts the user to new messages in his or her inbox.

1. In this lab, you will get first-hand experience with secondary tiles, push notifications, and toasts by adding them to Contoso Cookbook. At the conclusion, users will be able to pin favorite recipes to the start screen with secondary tiles, see tiles updated by the Windows Notification Service, and see scheduled toasts in action.

# Objectives

* 1. This lab will show you how to:
  + Create secondary tiles.
  + Use push notifications to update primary tiles.
  + Schedule toasts to message the user even if your app isn’t running.

# System requirements

* 1. You must have the following items to complete this lab:
  + Windows 8
  + Microsoft Visual Studio 2012

# Setup

* 1. To prepare your computer for this lab, you must:
  2. Install Windows 8.
  3. Install Microsoft Visual Studio 2012.

# Exercises

* 1. This hands-on lab includes the following exercises:
  2. Incorporate secondary tiles
  3. Incorporate push notifications
  4. Incorporate scheduled toasts
  5. Estimated time to complete this lab: **30 to 40 minutes**.

Exercise 1: Incorporate secondary tiles

1. In this exercise, you’ll add a command to the item-detail page’s app bar to enable users to pin favorite recipes to the start screen with secondary tiles. You’ll also add logic to Contoso Cookbook to show the corresponding recipe when the app is activated from a secondary tile.

Task 1 – Modify the app bar

* 1. The first thing we need to do is add a Pin command to the app bar. We then write a handler for it that creates a secondary tile, which shows the currently displayed recipe.
  2. Open the ContosoCookbook project you finished in Lab 6 in Visual Studio. If you didn’t complete Lab 6 or would like to start with a reference copy, you’ll find a completed version of the lab in the starting materials.
  3. Open ItemDetailPage.xaml and add the following statement to the end of the “LeftCommands” portion of the app bar.
     1. XAML
     2. <Button x:Name="PinRecipeButton" HorizontalAlignment="Left" Style="{StaticResource PinAppBarButtonStyle}" Click="OnPinRecipeButtonClicked" />
  4. Open StandardStyles.xaml and locate the commented-out **Style** element with key **PinAppBarButtonStyle**. Uncomment it so the resulting code looks like the following.
     1. XAML

<!--

<Style x:Key="RenameAppBarButtonStyle" TargetType="ButtonBase"   
 BasedOn="{StaticResource AppBarButtonStyle}">

<Setter Property="AutomationProperties.AutomationId" Value="RenameAppBarButton"/>

<Setter Property="AutomationProperties.Name" Value="Rename"/>

<Setter Property="Content" Value="&#xE13E;"/>

</Style> -->

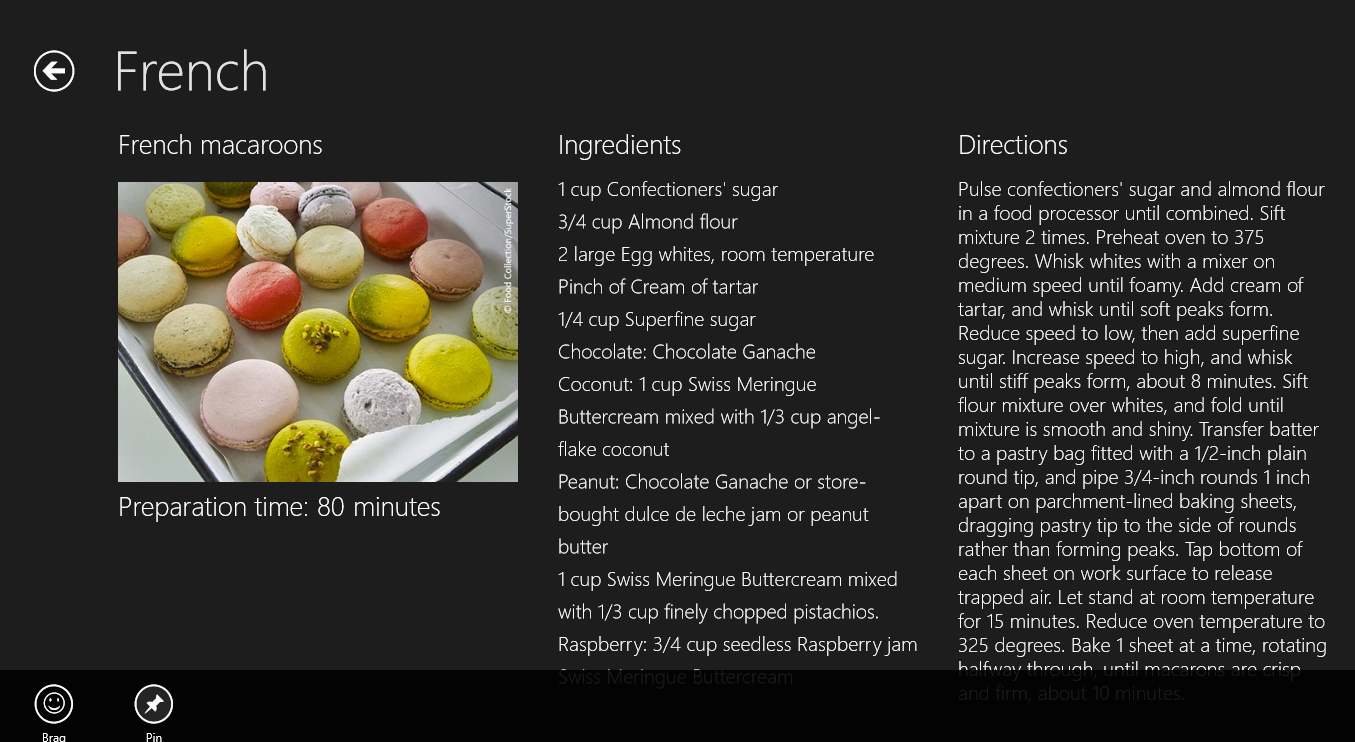
<Style x:Key="PinAppBarButtonStyle" TargetType="ButtonBase"   
 BasedOn="{StaticResource AppBarButtonStyle}">

<Setter Property="AutomationProperties.AutomationId" Value="PinAppBarButton"/>

<Setter Property="AutomationProperties.Name" Value="Pin"/>

<Setter Property="Content" Value="&#xE141;"/>

</Style>

* + 1. <!--  
       <Style x:Key="MusicInfoAppBarButtonStyle" TargetType="ButtonBase" BasedOn="{StaticResource AppBarButtonStyle}">
  1. Open ItemDetailPage.xaml.cs and add the following using statement.
     1. C#
     2. using Windows.UI.StartScreen;
  2. Now add the following method to the ItemDetailPage class.
     1. C#
     2. private async void OnPinRecipeButtonClicked(object sender, RoutedEventArgs e)
     3. {
     4. var item = (RecipeDataItem)this.flipView.SelectedItem;
     5. var uri = new Uri(item.TileImagePath.AbsoluteUri);
     6. var tile = new SecondaryTile(
     7. item.UniqueId, // Tile ID
     8. item.ShortTitle, // Tile short name
     9. item.Title, // Tile display name
     10. item.UniqueId, // Activation argument
     11. TileOptions.ShowNameOnLogo, // Tile options
     12. uri // Tile logo URI
     13. );
     14. await tile.RequestCreateAsync();
     15. }
     16. **Note:** The tile logo URIs in this step reference 150 x 150-pixel image files in the Images folder. Secondary tile images must be loaded locally. They can’t be loaded remotely like ordinary images can.
  3. Launch the app and tap a recipe to go to the item-detail page.
  4. Confirm that the app bar includes a **Pin** command, as shown in Figure 1.
     1. 
     2. Figure
     3. The Pin command
  5. Return to Visual Studio and stop debugging.

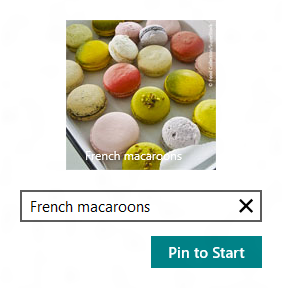
Task 2 – Modify OnLaunched

* 1. It’s easy to create a secondary tile, but you also need to recognize when the app was activated from a secondary tile, and navigate to the item-detail page to display a recipe.
  2. Open App.xaml.cs and find the OnLaunched method.
  3. Add the following statement before the **if(rootFrame == null)** clause.
     1. C#
     2. if (args.PreviousExecutionState == ApplicationExecutionState.Running)
     3. {
     4. if (!String.IsNullOrEmpty(args.Arguments))
     5. ((Frame)Window.Current.Content).Navigate(typeof(ItemDetailPage), args.Arguments);
     6. Window.Current.Activate();
     7. return;
     8. }
     9. **Note:** When a Windows Store app is started, **LaunchActivatedEventArgs.Kind** tells you *why* it was started, and **LaunchActivatedEventArgs.PreviousExecutionState** tells you whether the app was terminated after its previous run. Similarly, when an app is started from a secondary tile, **LaunchActivatedEventArgs.Arguments** provides the activation arguments passed in the fourth parameter to **SecondaryTile**’s constructor. What you’re doing here is navigating to the item-detail page to show a recipe, if the app was already running when the user tapped the secondary tile.
  4. Add the following statements to the **OnLaunched** method after the listeners for **SuggestionsRequested** and **CommandsRequested** events that we wired in previous labs. These statements navigate to the recipe page if the app was started from a secondary tile and was *not* already running.
     1. C#

// If the app was activated from a secondary tile, show the recipe

* + 1. if (!String.IsNullOrEmpty(args.Arguments))
    2. {
    3. rootFrame.Navigate(typeof(ItemDetailPage), args.Arguments);
    4. Window.Current.Content = rootFrame;
    5. Window.Current.Activate();
    6. return;
    7. }

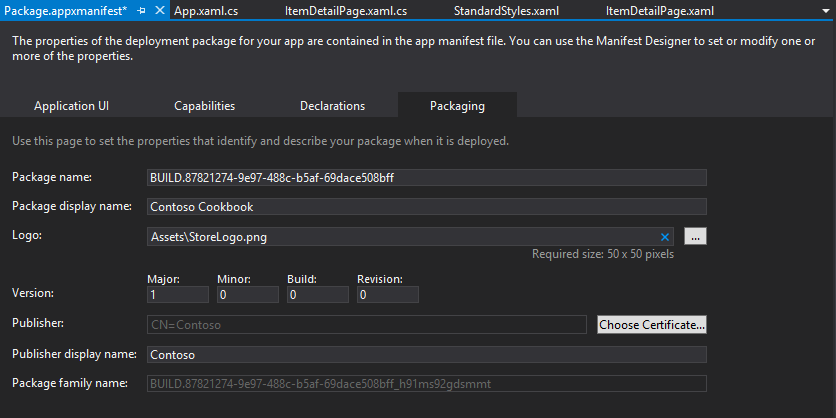
Task 3 – Pin a recipe

* 1. Now let’s test your modifications by pinning a recipe to the start screen and making sure that the correct recipe is shown when the app is launched from the tile.
  2. Press F5 to launch the app in the debugger.
  3. Go to the item-detail page to view the recipe of your choice.
  4. Slide the app bar up from the bottom of the screen (or press Windows logo key+Z) and tap the **Pin** button.
  5. Tap the **Pin to Start** button in the ensuing pop-up window (Figure 2).
     1. 
     2. Figure
     3. Pinning a secondary tile
  6. Return to Visual Studio and stop debugging.
  7. Go to the Windows 8 Start screen and verify that it now contains a secondary tile for the recipe that you pinned.
  8. Tap the secondary tile and verify that Contoso Cookbook starts up and displays the corresponding recipe.
  9. Press Alt+F4 to close the app.

Exercise 2: Incorporate push notifications

1. Push notifications enable Windows Store apps to update the content of their tiles – primary or secondary – even when the apps aren’t running. Push notifications can also update a tile’s *badge*, which is a number from 0 to 99 or a special glyph such as an asterisk that appears on the tile. Push notifications emanate from the Windows Notification Service (WNS) in response to calls from cloud-based services associated with your app. In this exercise, you’ll add push notifications to Contoso Cookbook.

Task 1 – Modify the app manifest

* 1. Microsoft has already deployed a Windows Azure service that Contoso Cookbook can call to subscribe to push notifications, and that transmits notifications through the WNS. When the service was deployed, it was configured to transmit notifications to apps that have a particular package name from a particular publisher (“Contoso”). In order to receive push notifications sent by the service, you need to modify the app manifest to use the correct package name and publisher ID. This is not something you normally need to do for your app, but because we want to reuse the Azure service that Microsoft has set up, we have to use the service’s package data.
  2. In Solution Explorer, double-click Package.appxmanifest to open the manifest.
  3. In the manifest editor, click the **Packaging** tab.
  4. Change **Package name** to “BUILD.87821274-9e97-488c-b5af-69dace508bff”.
  5. Change **Package display name** to “Contoso Cookbook.”
  6. Change **Publisher display name** to “Contoso.”
  7. Click the **Choose Certificate** button on the **Publisher** line. Then select **Select from file** from the **Configure Certificate** drop-down list and select Contoso.pfx from the ensuing dialog. You’ll find Contoso.pfx in the Certs folder of the starting materials. Then click **OK**.
  8. Verify that **Package name**, **Package display name**, **Publisher**, and **Publisher display name** have the values shown in Figure 3.
     1. 
     2. Figure 3
     3. The edited manifest
  9. Save your changes and close Package.appxmanifest.

Task 2 – Subscribe to push notifications

* 1. To subscribe to push notifications, a Windows Store app retrieves a notification channel from the Windows Runtime and passes the URI of the channel to a service. When the service wants to update a tile, it calls the WNS using the channel URI. The WNS then delivers the notification to the app. Let’s grab a channel URI when Contoso Cookbook starts up and use it to subscribe to push notifications.
  2. Open App.xaml.cs and add the following using statements.
     1. C#
     2. using Windows.UI.Notifications;
     3. using Windows.Networking.PushNotifications;
     4. using Windows.Security.Cryptography;
     5. using System.Net.Http;
     6. using Windows.Networking.Connectivity;
     7. using Windows.UI.Popups;
  3. Add the following statements to the **OnLaunched** method after the statement that registers a handler for **CommandsRequested** events.
     1. C#
     2. // Clear tiles and badges
     3. TileUpdateManager.CreateTileUpdaterForApplication().Clear();
     4. BadgeUpdateManager.CreateBadgeUpdaterForApplication().Clear();
     5. // Register for push notifications
     6. var profile = NetworkInformation.GetInternetConnectionProfile();
     7. if (profile.GetNetworkConnectivityLevel() == NetworkConnectivityLevel.InternetAccess)
     8. {
     9. var channel = await PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync();
     10. var buffer = CryptographicBuffer.ConvertStringToBinary(channel.Uri, BinaryStringEncoding.Utf8);
     11. var uri = CryptographicBuffer.EncodeToBase64String(buffer);
     12. var client = new HttpClient();
     13. try
     14. {
     15. var response = await client.GetAsync(new Uri("http://ContosoRecipes8.cloudapp.net?uri=" + uri + "&type=tile"));
     16. if (!response.IsSuccessStatusCode)
     17. {
     18. var dialog = new MessageDialog("Unable to open push notification channel");
     19. dialog.ShowAsync();
     20. }
     21. }
     22. catch (HttpRequestException)
     23. {
     24. var dialog = new MessageDialog("Unable to open push notification channel");
     25. dialog.ShowAsync();
     26. }
     27. }
     28. **Note:** The call to **CreatePushNotificationChannelForApplicationAsync** requests a notification channel from the Windows Runtime. In this example, you’re base-64-encoding the channel URI so you can pass it to the recipe service in a query string. Base-64- encoding isn’t strictly necessary, but it guards against the possibility that the channel URI might include characters that require base-64 encoding for inclusion in query strings.
     29. You’re using **HttpClient**, which is part of the .NET Framework (the Windows Runtime doesn’t include HTTP networking support), to fire a call off to the recipe service hosted in Windows Azure and pass the channel URI. That service maintains a record of all the clients (URIs) that have subscribed to it, and fires notifications to those clients every 2 minutes for 20 minutes. Each channel URI that is passed to the service identifies a particular app running on a particular device.

Task 3 – Test push notifications

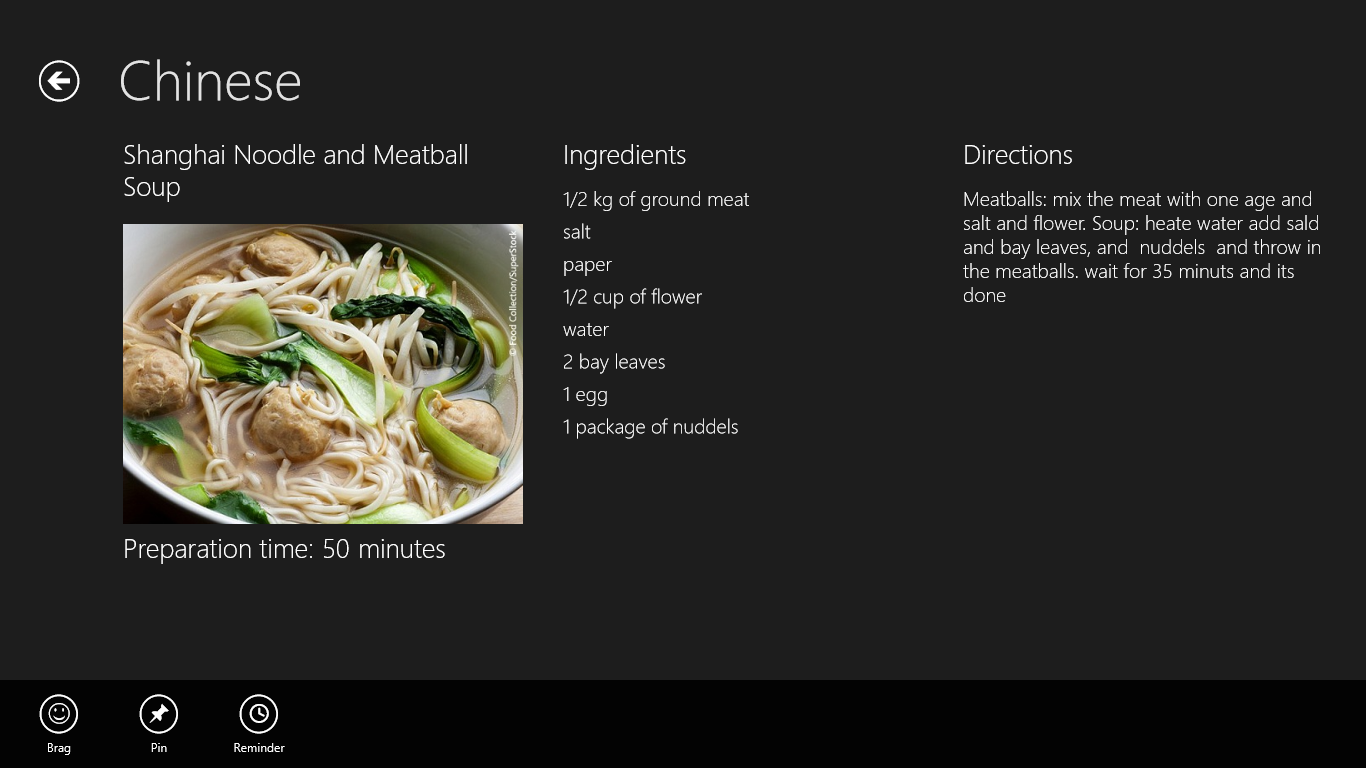
* 1. The final step is to make sure your app receives the push notifications transmitted to it.
  2. Press F5 to launch the app in the debugger. As part of its startup regimen, the app now resets its primary tile and subscribes to push notifications.
  3. Return to Visual Studio and stop debugging to close the app.
  4. Go to the Windows 8 Start screen and find the app’s primary tile. If it’s a square tile, right-click it and select “Larger” from the app bar to show a wide tile instead.
  5. Watch the tile for a few moments. Within two minutes, the tile should change to show one of several different featured recipes, as shown in Figure 4.
     1. 
     2. Figure 4
     3. Contoso Cookbook’s primary tile after a tile update
  6. Check back in another two minutes and verify that the tile has changed again. Note that all this is happening even though the app isn’t running!

**Note:** If you’d like to see badge notifications in action, change “type=tile” in the URI passed to **HttpClient.GetAsync** with “type=badge”. Run the app again and after about two minutes, a number, or “badge,” will appear on the tile.

Exercise 3: Incorporate scheduled toasts

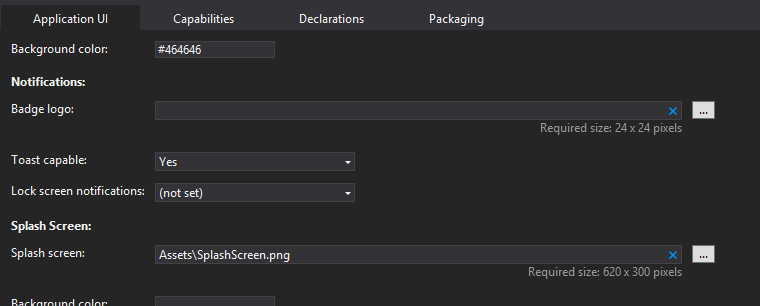
* 1. Toasts are messages that appear in small windows in the corner of the screen. Toasts appear regardless of whether or not the app that scheduled them is running. They can be scheduled by the app itself (while it’s running), or displayed in response to notifications from the Windows Notification Service. Moreover, they can include sounds as well as text.
  2. In this exercise, you’ll add a scheduled toast to Contoso Cookbook to simulate a reminder.

Task 1 – Modify the app bar

* 1. In order to provide a UI for scheduling a toast, we’ll add a **Reminder** command to the app bar and write a handler for it.
  2. Open ItemDetailPage.xaml and add the following statements to the <Page.Resources> section.
     1. XAML
     2. <Style x:Key="ReminderAppBarButtonStyle" TargetType="Button" BasedOn="{StaticResource AppBarButtonStyle}">
     3. <Setter Property="AutomationProperties.AutomationId" Value="ReminderAppBarButton"/>
     4. <Setter Property="AutomationProperties.Name" Value="Reminder"/>
     5. <Setter Property="Content" Value="&#xE121;"/>
     6. </Style>
  3. Also in ItemDetailPage.xaml, add the following statement to the end of the “LeftCommands” portion of the app bar.
     1. XAML
     2. <Button x:Name="ReminderButton" HorizontalAlignment="Left" Style="{StaticResource ReminderAppBarButtonStyle}" Click="OnReminderButtonClicked" />
  4. Open ItemDetailPage.xaml.cs and add the following using statements.
     1. C#
     2. using Windows.UI.Notifications;
     3. using Windows.UI.Popups;
  5. Next, add the following method to the ItemDetailPage class.
     1. C#
     2. private async void OnReminderButtonClicked(object sender, RoutedEventArgs e)
     3. {
     4. var notifier = ToastNotificationManager.CreateToastNotifier();
     5. // Make sure notifications are enabled
     6. if (notifier.Setting != NotificationSetting.Enabled)
     7. {
     8. var dialog = new MessageDialog("Notifications are currently disabled");
     9. await dialog.ShowAsync();
     10. return;
     11. }
     12. // Get a toast template and insert a text node containing a message
     13. var template = ToastNotificationManager.GetTemplateContent(ToastTemplateType.ToastText01);
     14. var element = template.GetElementsByTagName("text")[0];
     15. element.AppendChild(template.CreateTextNode("Reminder!"));
     16. // Schedule the toast to appear 30 seconds from now
     17. var date = DateTimeOffset.Now.AddSeconds(30);
     18. var stn = new ScheduledToastNotification(template, date);
     19. notifier.AddToSchedule(stn);
     20. }
  6. Start the app and tap a recipe to go to the item-detail page.
  7. Confirm that the app bar includes a **Reminder** command, as shown in Figure 5.
     1. 
     2. Figure 5
     3. The Reminder command
  8. Return to Visual Studio and stop debugging.

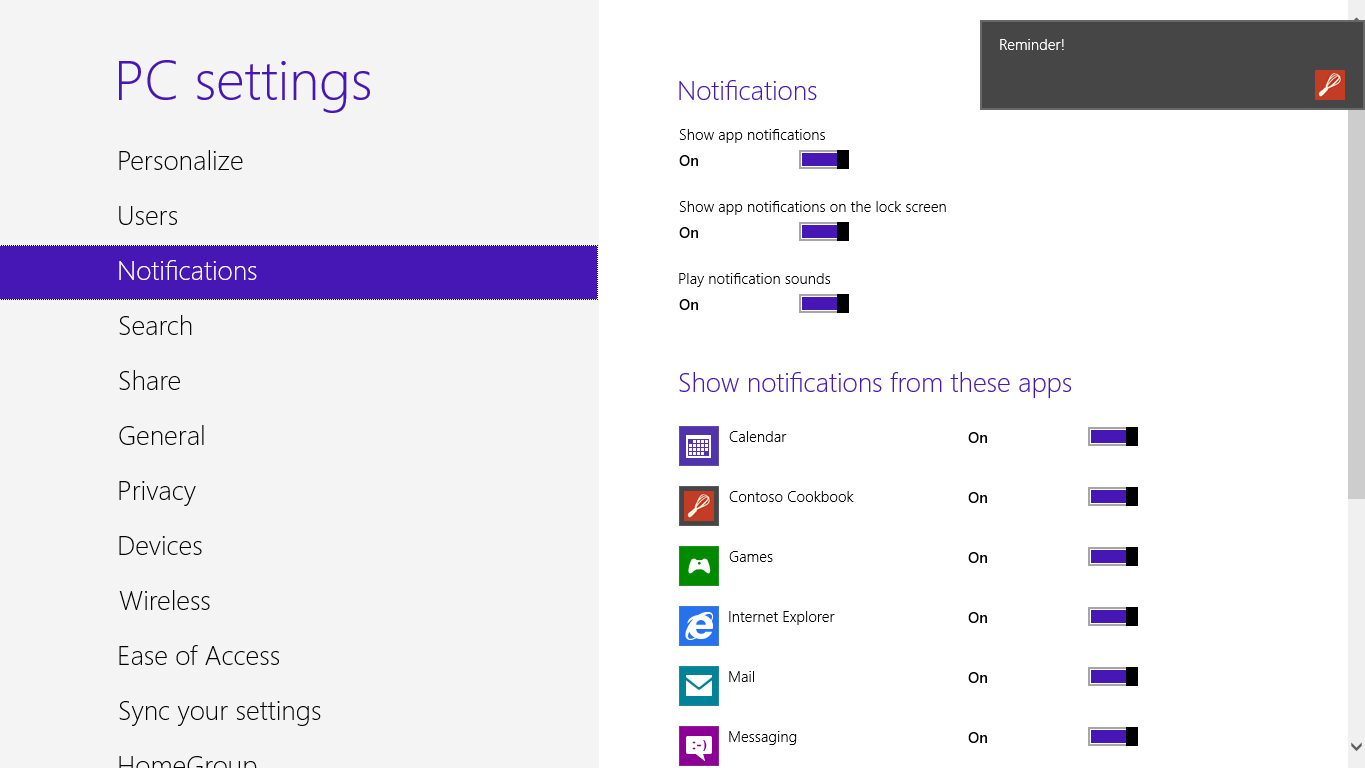
Task 2 – Enable toast notifications

* 1. One final task before we test is to enable toast notifications in the manifest.

1. Open Package.appxmanifest and go to the Application UI tab.
2. Change the “Toast capable” setting to “Yes” (Figure 6).
   * 1. 
     2. Figure 6
     3. Enabling toast notifications

**Note:** You must turn on **Toast capable** before an app can schedule toasts. After you enable this setting, a toggle button appears in the app’s permissions page that allows the user to turn notifications on and off. The **OnReminderButtonClicked** method you added checks to see if notifications are enabled and warns the user if they’re not.

Task 3 – Make a toast!

* 1. The final task is to test the code that schedules a toast notification, and see a toast in action.
  2. Press F5 to launch the app.
  3. Go to the recipe of your choice.
  4. Display the app bar and tap the **Reminder** button.
  5. Return to Visual Studio and stop debugging.
  6. Switch to the Windows 8 Start screen or to another app, and wait approximately 30 seconds for a toast to appear (Figure 7).
     1. 
     2. Figure 7
     3. A toast from Contoso Cookbook
  7. Tap the toast and verify that you switch back to Contoso Cookbook.
  8. Return to Visual Studio and stop debugging.

Summary

* 1. You’ve now seen secondary tiles, push notifications, and scheduled toasts in action. These are just a few examples of what you can do with tiles and notifications to update content when your app isn’t running.
  2. We’ve come a long way since we started on Contoso Cookbook back in Lab 1, but there’s more to do. Next up: exploring the Windows Store APIs and allowing a user to simulate a purchase of the app and of additional recipes. It’s extremely easy to monetize a Windows Store app, with Microsoft doing the bulk of the work to advertise it and collect purchase fees. Sound intriguing? Then let’s get to it!