Hands-On Lab

Tiles and Notifications

Lab version: 1.0.0

Last updated: 6/8/2012



Contents

[Overview 3](#_Toc319744252)

[Exercise 1: Incorporate Secondary Tiles 5](#_Toc319744253)

[Task 1 – Modify the Application Bar 5](#_Toc319744254)

[Task 2 – Modify OnLaunched 6](#_Toc319744255)

[Task 3 – Pin a Recipe 7](#_Toc319744256)

[Exercise 2: Incorporate Badge Notifications 8](#_Toc319744257)

[Task 1 – Modify the Application Manifest 8](#_Toc319744258)

[Task 2 – Subscribe to Badge Notifications 9](#_Toc319744259)

[Task 3 – Test Badge Notifications 11](#_Toc319744260)

[Exercise 3: Incorporate Scheduled Toasts 12](#_Toc319744261)

[Task 1 – Modify the Application Bar 12](#_Toc319744262)

[Task 2 – Make a Toast! 14](#_Toc319744263)

[Summary 15](#_Toc319744264)

Overview

* 1. Tiles are an important element of the Metro user experience. When an app is installed, a tile representing it is automatically created on the Metro 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 Metro project. That image measures 150 by 150 pixels and represents a *square tile*. However, you can enable *wide tiles* in your application by adding a 310-by-150 image to your project and designating it as the “wide logo” in the application manifest. If an application supports wide tiles, users can toggle between square tiles and wide tiles on the Metro start screen.
  2. Metro applications have the option of creating additional tiles known as *secondary tiles*. Secondary tiles serve as shortcuts to start up an application and have it go to a predefined location. For example, a weather application might allow users to create secondary tiles representing geographic locations such as Redmond, WA, and Atlanta, GA. A user may then start the application and view the weather in Redmond or Atlanta simply by tapping the secondary tile.
  3. Secondary tiles are created using a Windows Runtime API encapsulated in the Windows.UI.StartScreen.SecondaryTile class. The creation process is normally initiated by user input, such as tapping a command in the application bar. An application creating 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 representing activation arguments. When the app is launched from the secondary tile, the activation arguments are passed to it by the operating system. This provides the application with the information it needs to assume the state represented by the tile. For a weather application, the activation arguments might be nothing more than a zip code.
  4. Once a tile is created, its content doesn’t have to remain static. Metro-style apps enjoy a number of ways to make a tile’s content both dynamic and compelling via *tile updates* and *tile notifications*. For example, an application can use the Windows.UI.Notifications.TileUpdater class to update the contents of a primary or secondary tile while the application is running, and it can optionally queue updates to be shown even when it’s not running. (If queuing is enabled and an application adds updates to the queue, the operating system rotates between queued updates every few seconds, even if the application itself is no longer running. Up to five updates can be queued at any given time.)
  5. Perhaps more compelling is the ability for a tile to be updated remotely using notifications transmitted by the Windows Notification Service, or WNS. Suppose a weather application wanted to display severe-weather alerts on secondary tiles, even when the application isn’t running. The application can request a notification channel from the Metro runtime and pass the channel URI to a weather service running in the cloud. To warn the user about severe weather, the weather service calls the WNS using the channel URI provided to it, and the WNS, in turn, fires an asynchronous tile notification down to the device that registered the notification channel. The Metro runtime receives the notification and updates the tile, regardless of whether the weather application is currently running.
  6. Tile notifications aren’t the only type of notifications that can be triggered through the WNS. An application can also receive *toast notifications*, which pop up a toast window containing a message for the user (for example, "Severe weather detected in your area”), and *badge notifications*, which update a count shown on the face of the tile or optionally display a predefined glyph.

1. In this lab, you’ll get first-hand experience with tiles, badge notifications, and toasts by adding them to Contoso Cookbook. At the conclusion, users will be able to pin favorite recipes to the Metro start screen with secondary tiles, and to see badge updates from an Azure service indicating that new recipes are available.

# Objectives

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

# System Requirements

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

# Setup

* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install the Microsoft Windows 8 Release Preview
  3. Install the Microsoft Visual Studio 2012 RC for Windows 8

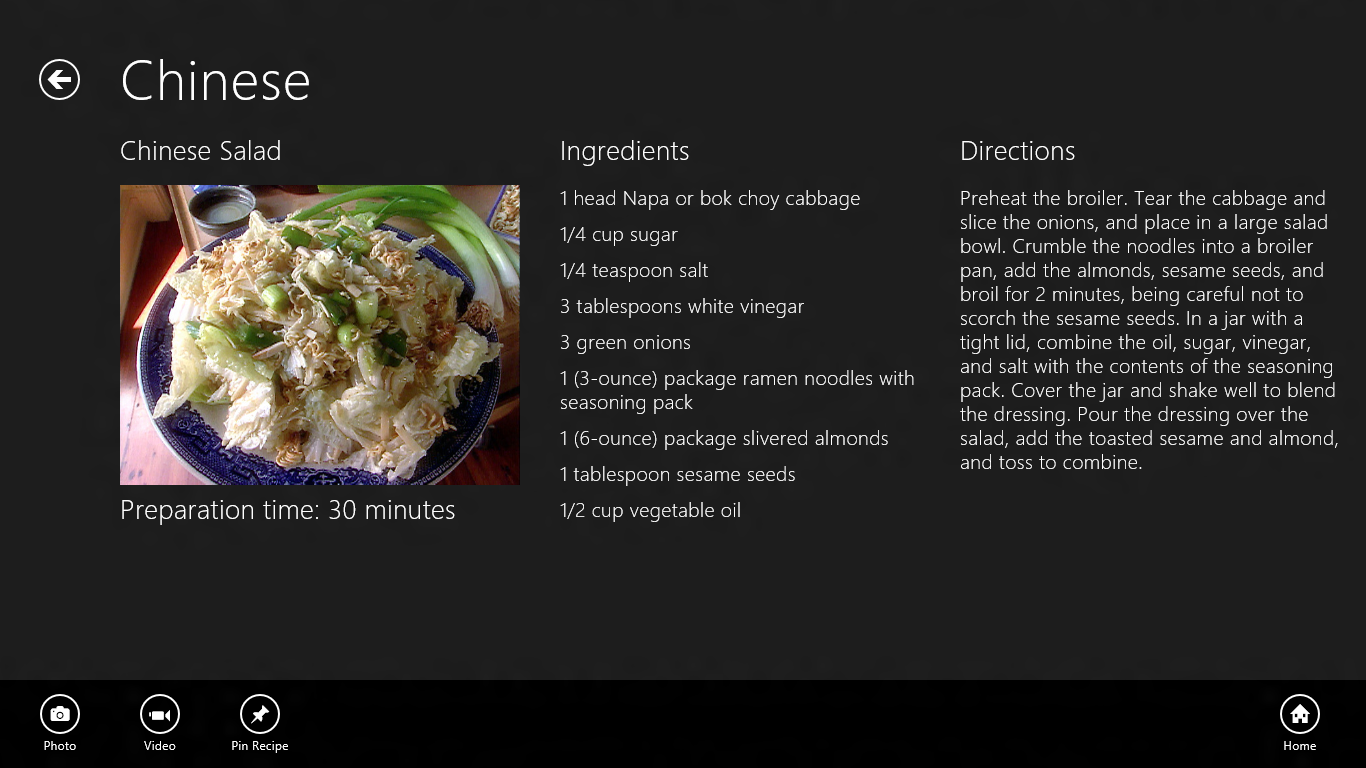
# Exercises

* 1. This Hands-On Lab comprises the following exercises:
  2. Incorporate Secondary Tiles
  3. Incorporate Badge 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 application bar enabling 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 Application Bar

* 1. The first thing we need to do is add a Pin Recipe command to the application bar, and write a handler for it that creates a secondary tile representing 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 statements to the <Page.Resources> section:
     1. XAML
     2. <Style x:Key="PinAppBarButtonStyle" TargetType="Button" BasedOn="{StaticResource AppBarButtonStyle}">
     3. <Setter Property="AutomationProperties.AutomationId" Value="PinAppBarButton"/>
     4. <Setter Property="AutomationProperties.Name" Value="Pin Recipe"/>
     5. <Setter Property="Content" Value="&#xE141;"/>
     6. </Style>
  4. Also in ItemDetailPage.xaml, add the following statement to the end of the “LeftCommands” portion of the application bar:
     1. XAML
     2. <Button x:Name="PinRecipe" HorizontalAlignment="Left" Style="{StaticResource PinAppBarButtonStyle}" Click="OnPinRecipeButtonClicked" />
  5. Open ItemDetailPage.xaml.cs and add the following using statement:
     1. C#
     2. using Windows.UI.StartScreen;
  6. Now add the following method to the ItemDetailPage class:
     1. C#
     2. private void OnPinRecipeButtonClicked(object sender, RoutedEventArgs e)
     3. {
     4. var logo = new Uri("ms-appx:///Assets/Logo.png");
     5. var tile = new SecondaryTile(
     6. \_item.UniqueId, // Tile ID
     7. \_item.ShortTitle, // Tile short name
     8. \_item.Title, // Tile display name
     9. \_item.UniqueId, // Activation argument
     10. TileOptions.ShowNameOnLogo, // Tile options
     11. logo // Tile logo
     12. );
     13. tile.RequestCreateAsync();
     14. }
  7. Launch the application and tap a recipe to go to the item-detail page.
  8. Confirm that the application bar includes a Pin Recipe command, as shown in Figure 1.
  9. 
  10. Figure 1
  11. The Pin Recipe command
  12. Return to Visual Studio and stop debugging.

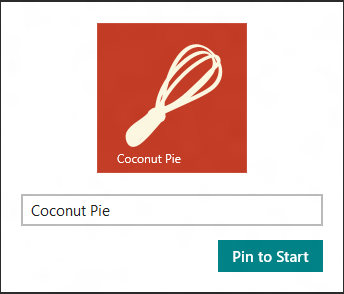
Task 2 – Modify OnLaunched

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.

* 1. Open App.xaml.cs and find the OnLaunched method.
  2. Add the highlighted if clause to the method, placing the remaining statements (which are already present in OnLaunched) into an else clause:
     1. C#
     2. // Create a Frame to act as the navigation context and associate it with
     3. // a SuspensionManager key
     4. var rootFrame = new Frame();
     5. SuspensionManager.RegisterFrame(rootFrame, "AppFrame");
     6. await RecipeDataSource.LoadRemoteDataAsync();
     7. if (args.Arguments.Equals(string.Empty))
     8. {
     9. // Find out if the "Remember where I was" preference is enabled
     10. bool remember = false;
     11. if (ApplicationData.Current.RoamingSettings.Values.ContainsKey("Remember"))
     12. remember = (bool)ApplicationData.Current.RoamingSettings.Values["Remember"];
     13. if (args.PreviousExecutionState == ApplicationExecutionState.Terminated || remember)
     14. {
     15. // Restore the saved session state only when appropriate
     16. await SuspensionManager.RestoreAsync();
     17. }
     18. if (rootFrame.Content == null)
     19. {
     20. // When the navigation stack isn't restored navigate to the first page,
     21. // configuring the new page by passing required information as a navigation
     22. // parameter
     23. if (!rootFrame.Navigate(typeof(GroupedItemsPage), "AllGroups"))
     24. {
     25. throw new Exception("Failed to create initial page");
     26. }
     27. }
     28. }
     29. else
     30. {
     31. rootFrame.Navigate(typeof(ItemDetailPage), args.Arguments.ToString());
     32. }
     33. // Place the frame in the current Window and ensure that it is active
     34. Window.Current.Content = rootFrame;
     35. Window.Current.Activate();
     36. **Note:** When a Metro application is launched, LaunchActivatedEventArgs.Kind tells you *why* it was launched, and LaunchActivatedEventArgs.PreviousExecutionState tells you whether the application was terminated after its previous run. Similarly, when an application is launched from a secondary tile, LaunchActivatedEventArgs.Arguments provides the activation arguments passed in the fourth parameter to SecondaryTile’s constructor. What you’re doing is storing the arguments string so it can be used to navigate to a particular recipe once the recipe data is loaded, and you’re affording tile activation precedence over the “Remember where I was” feature you added in Lab 6, and over the PLM logic you added in Lab 5.

Task 3 – Pin a Recipe

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 application is launched from the tile.

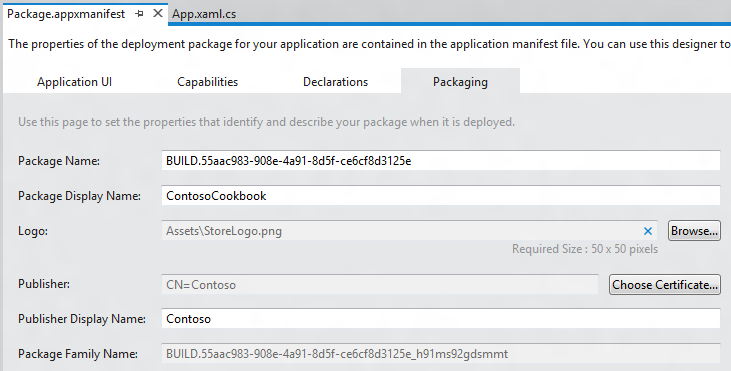
* 1. Press F5 to launch the application in the debugger.
  2. Go to the item-detail page to view the recipe of your choice.
  3. Slide the application bar up from the bottom of the screen (or press Win-Z) and tap the Pin Recipe button.
  4. Tap the Pin to Start button in the ensuing popup (Figure 2).
  5. 
  6. Figure 2
  7. Pinning a secondary tile
  8. Return to Visual Studio and stop debugging.
  9. Go to the Metro start screen and verify that it now contains a secondary tile representing the recipe that you pinned.
  10. Tap the secondary tile and verify that Contoso Cookbook starts up and displays the corresponding recipe.
  11. Return to Visual Studio and stop debugging.

Exercise 2: Incorporate Badge Notifications

1. Tile notifications enable Metro applications to update the content of their tiles – primary or secondary – even when the applications aren’t running. Badge notifications are a special type of tile notification that update a tile’s *badge*, which is a number from 0 to 99 or a special glyph such as an asterisk that appears in a predetermined location on the tile. Badge notifications emanate from the Windows Notification Service (WNS) in response to calls from cloud-based services associated with your application. In this exercise, you’ll add a simple badge notification to Contoso Cookbook allowing you to inform the user when new recipes are available.

Task 1 – Modify the Application Manifest

Microsoft has already deployed a Windows Azure service that Contoso Cookbook can call to subscribe to badge notifications, and that transmits notifications through the WNS. When the service was deployed, it was configured to transmit notifications to applications possessing a particular package name from a particular publisher (“Contoso”). In order to receive badge notifications sent by the service, you need to modify the application manifest to use the correct package name and publisher ID.

* 1. In Solution Explorer, double-click Package.appxmanifest to open the manifest.
  2. In the manifest editor, click the Packaging tab.
  3. Change Package Name to “BUILD.55aac983-908e-4a91-8d5f-ce6cf8d3125e”.
  4. Change Publisher Display Name to “Contoso.”
  5. Click the “Choose Certificate” button on the Publisher line. Then select “Select from file…” from the “Configure Certificate” drop-down and select Contoso.pfx from the ensuing dialog. You’ll find Contoso.pfx in the certs folder of the starting materials. Then click OK.
  6. Verify that Package Name, Package Display Name, Publisher, and Publisher Display Name have the values shown in Figure 3.
  7. 
  8. Figure 3
  9. The edited manifest
  10. Save your changes and close Package.appxmanifest.

Task 2 – Subscribe to Badge Notifications

To subscribe to badge notifications, a Metro application retrieves a notification channel from the Metro runtime and passes the URI of the channel to a service. When the service wishes to update a badge, it calls the WNS using the channel URI, and the WNS delivers a badge notification to the application. Let’s grab a channel URI when Contoso Cookbook starts up and use it to subscribe to badge notifications.

* 1. 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.UI.Popups;
  2. Now find the OnLaunched method and add an async keyword to the method declaration. This is necessary because in the next step, you’ll add code that uses C#’s await keyword:
     1. C#
     2. protected async override void OnLaunched(LaunchActivatedEventArgs args)
  3. Add the following statements to the end of the OnLaunched method to subscribe to badge notifications each time the application starts up. Be sure to add it to the end to prevent other startup tasks from being executed if the network is down or slow to respond:
     1. C#
     2. // Register for badge notifications
     3. var channel = await PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync();
     4. var buffer = CryptographicBuffer.ConvertStringToBinary(channel.Uri, BinaryStringEncoding.Utf8);
     5. var uri = CryptographicBuffer.EncodeToBase64String(buffer);
     6. var client = new HttpClient();
     7. try
     8. {
     9. var response = await client.GetAsync(new Uri("http://ContosoRecipes8.cloudapp.net?uri=" + uri));
     10. if (!response.IsSuccessStatusCode)
     11. {
     12. var dialog = new MessageDialog("Unable to open push notification channel");
     13. dialog.ShowAsync();
     14. }
     15. }
     16. catch (HttpRequestException)
     17. {
     18. var dialog = new MessageDialog("Unable to open push notification channel");
     19. dialog.ShowAsync();
     20. }
     21. **Note:** The call to CreatePushNotificationChannelForApplicationAsync requests a notification channel from the Metro 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.
     22. You’re using HttpClient, which is part of .NET (WinRT 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 badge notifications to those clients every 60 seconds for one hour. Each channel URI passed to the service identifies a particular application running on a particular device. The service “dedupes” channel URIs so that Contoso Cookbook doesn’t receive multiple badge updates if it passes the same channel URI multiple times. If it receives the same channel URI again, the service “refreshes” it so that the corresponding application will receive badge notifications for another 60 minutes.
  4. Add the following statements before the ones you added in the previous step to clear badge notifications each time the application starts up:
     1. C#
     2. // Clear the badge
     3. BadgeUpdateManager.CreateBadgeUpdaterForApplication().Clear();
     4. **Note:** Once your app starts receiving badge notifications, a number will appear on the application’s primary tile. By default, that number, or *badge*, never goes away. The statement you just added uses the Windows Runtime’s BadgeUpdater.Clear method to clear the badge from the tile. Since the application doesn’t have to be running for badge notifications to be received, you’ll test the badge notifications without the application running, and to clear the badge, all you have to do is run the application again.

Task 3 – Test Badge Notifications

The final step is to make sure your application receives the badge notifications transmitted to it.

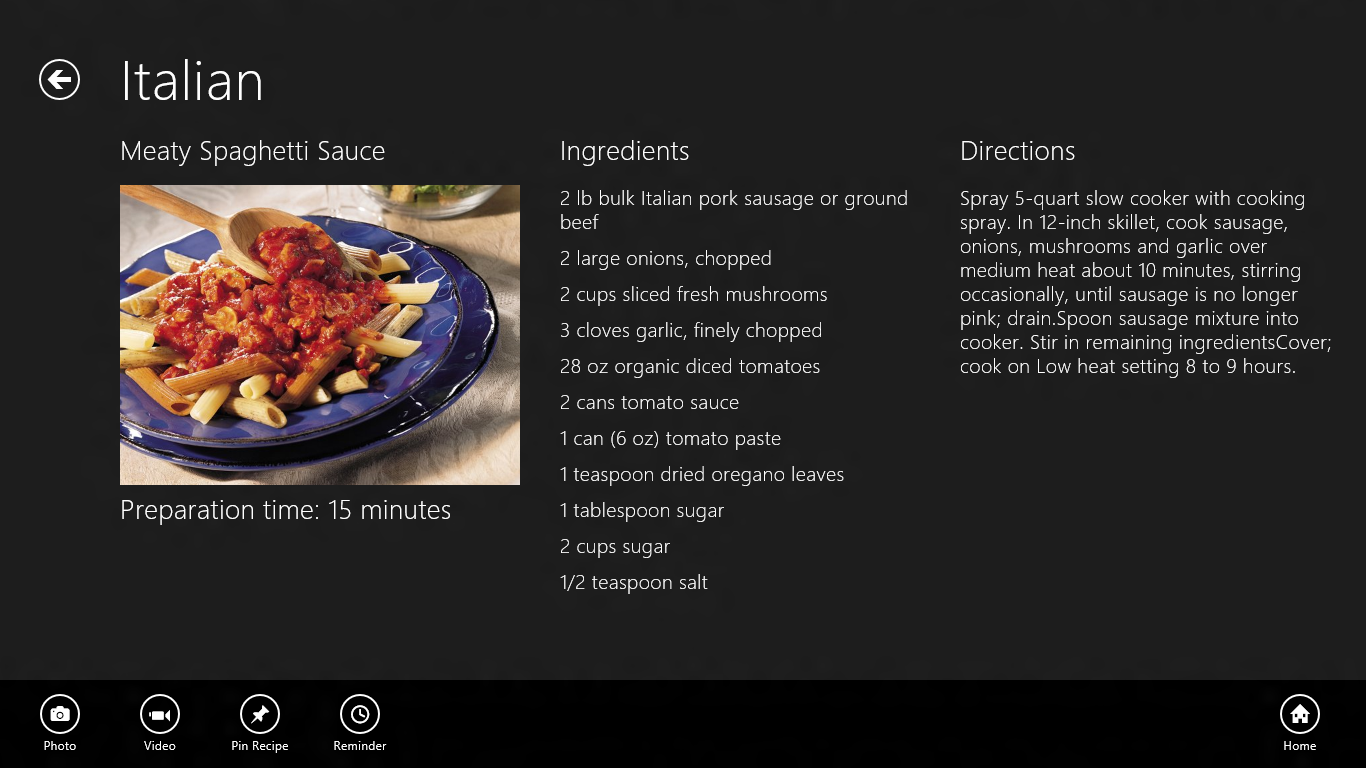
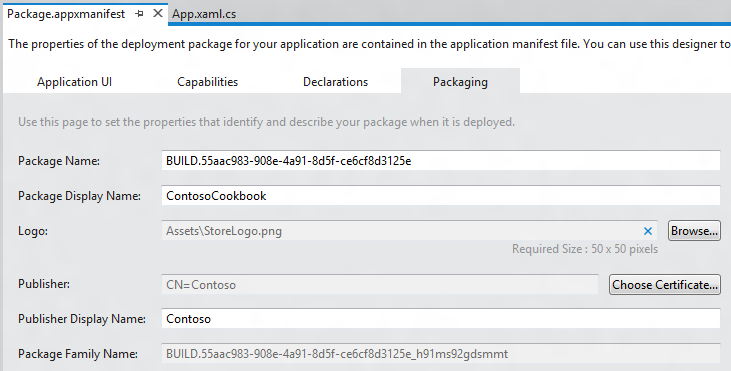
* 1. Press F5 to launch the application in the debugger. As part of its startup regimen, the application now subscribes to badge notifications.
  2. Return to Visual Studio and stop debugging to close the application.
  3. Go to the Metro start screen and find the application’s primary tile.
  4. Watch the tile for a few moments. Within 60 seconds, a number should appear on it, as shown in Figure 4. Conceptually, that number represents the number of new recipes available. In reality, it is simply the number of minutes after the hour.
  5. 
  6. Figure 4
  7. The primary tile showing a badge
  8. Continue watching the tile for two or three minutes and verify that the badge increments every 60 seconds.
  9. Run the application again from Visual Studio.
  10. Return to the Metro start screen and confirm that the badge has disappeared from the tile. (It will reappear again sometime in the next 60 seconds, but the fact that it’s gone indicates that the call to BadgeUpdater.Clear executed each time the application starts up is working.)
  11. Return to Visual Studio and stop debugging.

Exercise 3: Incorporate Scheduled Toasts

* 1. Toasts are messages that appear in “toast” windows in the corner of the screen. Toasts appear regardless of whether or not the application that scheduled them is running, and they can be scheduled by the application 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 Application Bar

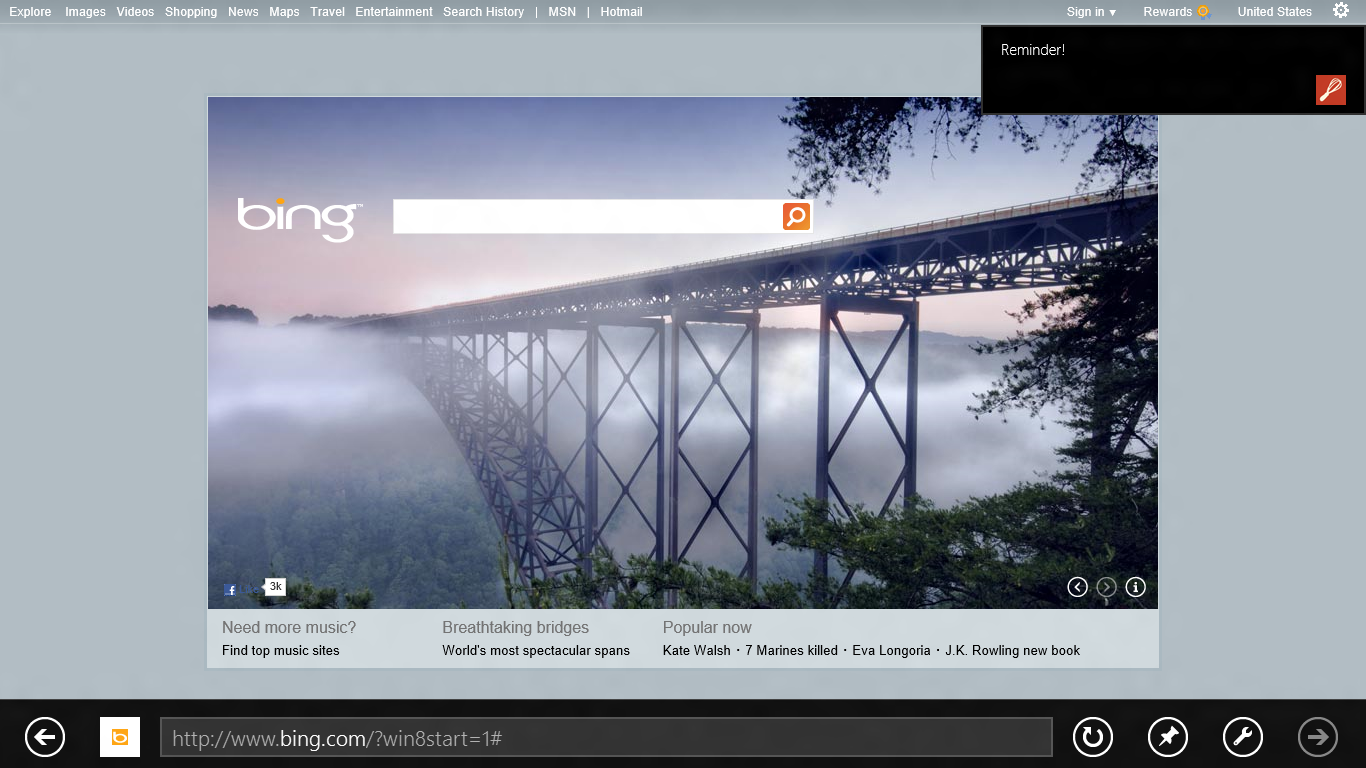
In order to provide a UI for scheduling a toast, we’ll add a Reminder command to the application bar and write a handler for it.

* 1. 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="PinAppBarButton"/>
     4. <Setter Property="AutomationProperties.Name" Value="Reminder"/>
     5. <Setter Property="Content" Value="&#xE121;"/>
     6. </Style>
  2. Also in ItemDetailPage.xaml, add the following statement to the end of the “LeftCommands” portion of the application bar:
     1. XAML
     2. <Button x:Name="Reminder" HorizontalAlignment="Left" Style="{StaticResource ReminderAppBarButtonStyle}" Click="OnReminderButtonClicked" />
  3. Open ItemDetailPage.xaml.cs and add the following using statements:
     1. C#
     2. using Windows.UI.Notifications;
     3. using Windows.UI.Popups;
  4. Next, add the following method to the ItemDetailPage class:
     1. C#
     2. private 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. 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. }
  5. Launch the application and tap a recipe to go to the item-detail page.
  6. Confirm that the application bar includes a Reminder command, as shown in Figure 5.
  7. 
  8. Figure 5
  9. The Reminder command
  10. Return to Visual Studio and stop debugging.
  11. One final task before we test is to enable toast notifications in the manifest. Open Package.appxmanifest and change the “Toast Capable” setting to “Yes” on the Application UI tab (Figure 6).
  12. 
  13. Figure 6
  14. Enabling toast notifications

**Note:** “Toast Capable” must be turned on before an application can schedule toasts. Once this setting is enabled, a toggle button appears in the app’s permissions page allowing the user to turn notifications on and off. The OnRedminderButtonClicked method you added checks to see if notifications are enabled and warns the user if they’re not.

Task 2 – Make a Toast!

The final task is to test the code that schedules a toast notification and see a toast in action.

* 1. Press F5 to launch the application.
  2. Go to the recipe of your choice.
  3. Display the application bar and tap the “Reminder” button.
  4. Return to Visual Studio and stop debugging.
  5. Switch to the Metro start screen or to another app and wait approximately 30 seconds for a toast to appear (Figure 7).
  6. 
  7. Figure 7
  8. A toast from Contoso Cookbook
  9. Tap the toast and verify that you switch back to Contoso Cookbook.

Summary

* 1. You’ve now seen secondary tiles, badge notifications, and scheduled toasts in action. These are but a few examples of what you can do with tiles and notifications to update content when your application 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 application and of additional recipes. It’s extremely easy to monetize a Metro application, with Microsoft doing the bulk of the work to advertise it and collect purchase fees. Sound intriguing? Then let’s get to it!