

Hands-on lab

Lab 6: Settings and preferences

September 2012



Contents

[Overview 3](#_Toc334679826)

[Exercise 1: Add an About page 4](#_Toc334679827)

[Task 1 – Add an About command 4](#_Toc334679828)

[Task 2 – Add an About page 5](#_Toc334679829)

[Task 3 – Test the results 6](#_Toc334679830)

[Exercise 2: Add a preferences page 7](#_Toc334679831)

[Task 1 – Add a preferences command 7](#_Toc334679832)

[Task 2 – Add a preferences page 8](#_Toc334679833)

[Task 3 – Make the preference sticky 9](#_Toc334679834)

[Exercise 3: Implement the preference 11](#_Toc334679835)

[Task 1 – Modify the OnLaunched method 12](#_Toc334679836)

[Task 2 – Test the results 13](#_Toc334679837)

[Summary 14](#_Toc334679838)

Overview

* 1. Lab 3 introduced contracts and demonstrated how easily apps can integrate with the Share and Search contracts. The contracts also include a Settings charm, which changes settings in the active Windows Store app. In the settings pane that appears when you select the Settings charm, the operating system provides a Permissions command that allows users to enable and disable certain features of the program, such as webcam and microphone access. Significantly, you can add commands of your own to the settings pane and connect them to settings pages. This provides users with convenient access to preferences, about boxes, and other app-specific settings content.
  2. In this lab, you’ll add **About** and **Preferences** commands to the settings pane in Contoso Cookbook. You’ll expose a simple user preference that can be toggled on and off with a toggle switch, and you’ll use roaming settings to store that preference so it will follow users wherever they go.

# Objectives

* 1. This lab will show you how to:
  + Add an **About** command and an About page to the settings pane.
  + Add a **Preferences** command and a preferences page to the settings pane.
  + Use roaming settings to store user preferences.

# 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. Add an About page
  3. Add a preferences page
  4. Implement the preference
  5. Estimated time to complete this lab: **30 to 40 minutes**.

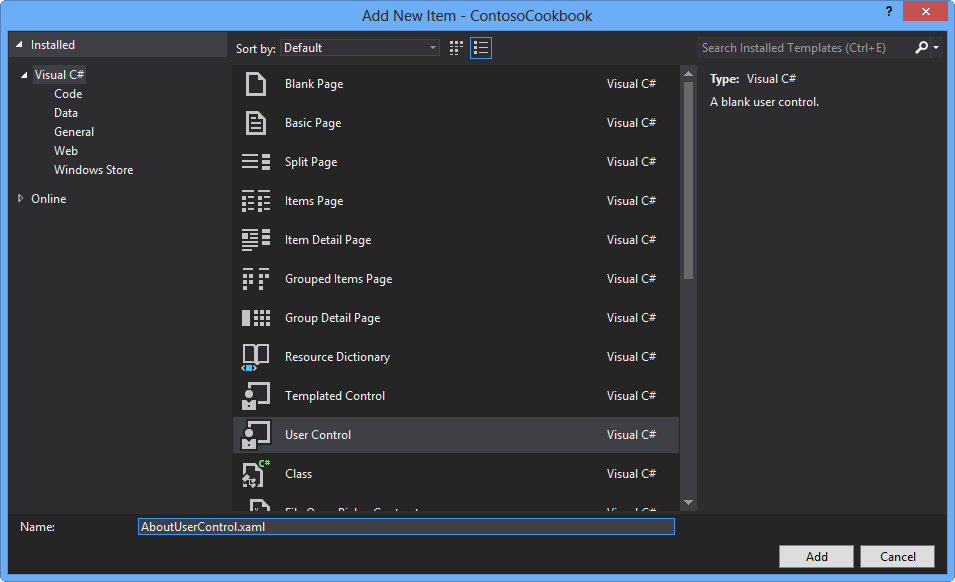
Exercise 1: Add an About page

1. In this exercise, you’ll add a simple About page to Contoso Cookbook. To host the About page, you’ll use the **SettingsFlyout** class in the Callisto library you added to the project in Lab 4.

Task 1 – Add an About command

* 1. The first step is to add an **About** command to the settings menu, which we will accomplish by handling **SettingsPane.CommandsRequested** events.
  2. Open the ContosoCookbook project you finished in Lab 4 in Visual Studio. If you didn’t complete Lab 4 or would like to start with a reference copy, you’ll find a completed version of the lab in the starting materials.
  3. Open App.xaml.cs and add the following using statements.
     1. C#
     2. using Windows.UI.ApplicationSettings;
     3. using Callisto.Controls;
     4. using Windows.UI;
  4. Add the following field to the **App** class.
     1. C#
     2. private Color \_background = Color.FromArgb(255, 0, 77, 96);
  5. Add the following statements to the **OnLaunched** method*,* right after the statement that registers a handler for **SuggestionsRequested** events.
     1. C#
     2. // Register handler for CommandsRequested events from the settings pane
     3. SettingsPane.GetForCurrentView().CommandsRequested += OnCommandsRequested;
  6. Add the same statements to the **OnSearchActivated** method (again, right after the statement that registers a handler for **SuggestionsRequested** events) to ensure **CommandsRequested** events get handled even if the app is activated from the Windows 8 search pane.
  7. Add the following event handler to App.xaml.cs.
     1. C#
     2. void OnCommandsRequested(SettingsPane sender, SettingsPaneCommandsRequestedEventArgs args)
     3. {
     4. // Add an About command
     5. var about = new SettingsCommand("about", "About", (handler) =>
     6. {
     7. var settings = new SettingsFlyout();
     8. settings.Content = new AboutUserControl();
     9. settings.HeaderBrush = new SolidColorBrush(\_background);
     10. settings.Background = new SolidColorBrush(\_background);
     11. settings.HeaderText = "About";
     12. settings.IsOpen = true;
     13. });
     15. args.Request.ApplicationCommands.Add(about);
     16. }
     17. **Note:** Here you add a command to the settings menu by adding a **SettingsCommand** object to the **ApplicationCommands** collection that is passed to **CommandsRequested** events. The third parameter for the **SettingsCommand** constructor is the handler that’s called when the command is invoked. In this example, you use Callisto’s **SettingsFlyout** class to display an About page from the handler. Of course, that page doesn’t exist yet. You’ll take care of that in the next task.

Task 2 – Add an About page

* 1. The event handler that you added in the previous task adds an **About** command to the settings pane. The next step is to add an about page for the **About** command to display. To create that page, we’ll add a new user control to the project.
  2. In Solution Explorer, right-click the project and use the **Add - New Item** command to add a user control to the project. Name the file AboutUserControl.xaml, as shown in Figure 1.
     1. 
     2. Figure 1
     3. Adding a user control representing the About page
  3. Add the following statements to the empty Grid in AboutUserControl.xaml.
     1. XAML
     2. <StackPanel>
     3. <TextBlock Text="Contoso Cookbook" FontFamily="Segoe UI" FontWeight="SemiLight" FontSize="26.667" />
     4. <TextBlock Text="Trial Version" FontFamily="Segoe UI" FontWeight="SemiLight" FontSize="18" />
     5. </StackPanel>
     6. **Note:** The About page currently informs the user that this is a trial version of Contoso Cookbook. In Lab 8, you’ll use the Windows Runtime’s store APIs to simulate purchases of the app, and after a purchase occurs, you’ll replace “Trial Version” with licensing information.

Task 3 – Test the results

* 1. Now it’s time to test your changes and see what an About page looks like.
  2. Press F5 to run the app.
  3. Display the charms and select **Settings**.
  4. Select the **About** command from the settings pane.
  5. Confirm that the About page appears, as shown in Figure 2.
     1. 
     2. Figure 2
     3. Contoso Cookbook’s About page
  6. Return to Visual Studio and stop debugging.

Exercise 2: Add a preferences page

1. Now that you understand the mechanics of adding a page to the settings pane, you’ll add another page – this time, a preferences page that allows the user to enter and edit preferences. We’ll add just one preference to demonstrate how it’s done, but of course you’re free to add as many preferences as you’d like. The preference you’ll add is one that allows the user to configure Contoso Cookbook to return, when it starts up, to the last recipe or recipe group that was displayed.

Task 1 – Add a preferences command

* 1. Start by modifying the **CommandsRequested** event handler you wrote in the previous exercise so that it adds a **Preferences** command, too.
  2. Open App.xaml.cs and find the **OnCommandsRequested** method.
  3. Add the following statements to **OnCommandsRequested** after the statement that adds an **About** command.
     1. C#
     2. // Add a Preferences command
     3. var preferences = new SettingsCommand("preferences", "Preferences", (handler) =>
     4. {
     5. var settings = new SettingsFlyout();
     6. settings.Content = new PreferencesUserControl();
     7. settings.HeaderBrush = new SolidColorBrush(\_background);
     8. settings.Background = new SolidColorBrush(\_background);
     9. settings.HeaderText = "Preferences";
     10. settings.IsOpen = true;
     11. });
     12. args.Request.ApplicationCommands.Add(preferences);

Task 2 – Add a preferences page

* 1. The next task is to create the page that the **Preferences** command invokes.
  2. In Solution Explorer, right-click the project and use the **Add > New Item** command to add a new user control to the project. Name the file PreferencesUserControl.xaml.
  3. Add the following statement to the empty Grid element in PreferencesUserControl.xaml.
     1. XAML
     2. <ToggleSwitch x:Name="Remember" Header="Remember where I was" />
  4. Press F5 to run the app.
  5. Display the charms and select the Settings charm.
  6. Select the **Preferences** command from the settings pane.
  7. Confirm that the preferences page appears and that it contains a toggle switch, as shown in Figure 3.
     1. 
     2. Figure 3
     3. Contoso Cookbook’s preferences page
  8. Return to Visual Studio and stop debugging.

Task 3 – Make the preference sticky

* 1. Right now, the “Remember where I was” toggle switch in the preferences page isn’t wired up to anything, and it doesn’t retain its state. Let’s fix that by using roaming settings to save the state of the toggle switch each time it’s clicked, and to initialize it each time the preferences page is displayed.
  2. Open PreferencesUserControl.xaml and add a **Toggled** attribute to the **ToggleSwitch** control.
     1. XAML
     2. <ToggleSwitch x:Name="Remember" Header="Remember where I was" Toggled="OnToggled" />
  3. Open PreferencesUserControl.xaml.cs and add the following using statement.
     1. C#
     2. using Windows.Storage;
  4. Then add the following method.
     1. C#
     2. private void OnToggled(object sender, RoutedEventArgs e)
     3. {
     4. ApplicationData.Current.RoamingSettings.Values["Remember"] = Remember.IsOn;
     5. }
  5. Add the following statements to the **PreferencesUserControl** constructor, after the call to **InitializeComponent**, to initialize the toggle switch each time the Preferences page is displayed.
     1. C#
     2. // Initialize the ToggleSwitch from roaming settings
     3. if (ApplicationData.Current.RoamingSettings.Values.ContainsKey("Remember"))

Remember.IsOn = (bool)ApplicationData.Current.RoamingSettings.Values["Remember"];

* + 1. **Note:** To assist with the task of saving and restoring settings and other app data, the Windows Runtime gives you the **Windows.Storage.ApplicationData** class. **ApplicationData** lets you store data locally, in the cloud (roaming storage), or in temporary storage. The data you save can be stored in the form of name-value pairs, through **ApplicationData** properties named **LocalSettings** and **RoamingSettings**. Or, it can be stored in files created in special app-specific folders that are accessed through **ApplicationData’s** **LocalFolder**, **RoamingFolder**, and **TemporaryFolder** properties.
    2. The benefit of persisting data in one of the roaming repositories is that such data “follows” the user from one device to another. Moreover, if an app writes data to **RoamingSettings** or **RoamingFolder** and the user isn’t signed in with a Microsoft account or doesn’t have an Internet connection, the Windows Runtime automatically persists the data locally. So there’s little to lose and lots to gain by using **RoamingSettings** or **RoamingFolder** as a store for user preferences. The only caveat is that the platform limits how much data can be saved in roaming storage. In Windows 8 Release Preview, you can roam about 100 kilobytes (KB) of data. You can determine at run time how much quota you have from the **ApplicationData.RoamingStorageQuota** property.
  1. Press F5 to run the app.
  2. Display the charms and select the Settings charm.
  3. Select the **Preferences** command from the settings pane.
  4. Tap **Remember where I was** to turn the toggle switch on.
  5. Dismiss the settings pane.
  6. Return to Visual Studio and stop debugging.
  7. Press F5 to start the app again.
  8. Go to the preferences page and confirm that the toggle switch is on.
  9. Return to Visual Studio and stop debugging.

Exercise 3: Implement the preference

1. Currently, Contoso Cookbook shows the start page each time it starts up. The purpose of adding a user preference in the previous exercise, called “Remember where I was”, was to allow a user to configure the app to return, each time it starts up, to the page that was displayed the last time it shut down. This user preference requires only minor changes to your code, since Visual Studio has already included code in your app that saves the navigation state when the app is suspended.
   * 1. **Note:** Process Lifetime Management, or PLM, is an important element of a Windows Store app. When an app is suspended, it can be terminated at any time by the operating system. And if the app is terminated, it loses its state.
     2. Users won’t care much for apps that lose their state just because they temporarily switched away from them. That’s why the **Windows.UI.Xaml.Application** class defines an event named **Suspending**. Just before an app is suspended, a **Suspending** event fires. This provides the app with the opportunity to save its state, in case it is terminated by the operating system and later reactivated by the user. The goal is to restore that state when the user reactivates the app, to create the illusion that it was never terminated at all.
     3. Visual Studio included a class named **SuspensionManager** in your app. It is located in SuspensionManager.cs in the project’s Common folder. Visual Studio also included a line of code in the **App** constructor in App.xaml.cs, which registers a handler for **Suspending** events. The handler—**OnSuspending**—calls **SuspensionManager.SaveAsync** to save the app’s navigation state. *Navigation state* includes the item or group the user was viewing, as well as the route by which the user arrived at that item or group.

private async void OnSuspending(object sender, SuspendingEventArgs e)

{

var deferral = e.SuspendingOperation.GetDeferral();

await SuspensionManager.SaveAsync();

deferral.Complete();

}

* + 1. In addition, Visual Studio included an **if** clause in the **OnLaunched** method in App.xaml.cs that restores the app’s navigation state if the app was terminated by the operating system after it was suspended:

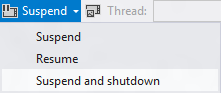
if (args.PreviousExecutionState == ApplicationExecutionState.Terminated)

{

// Restore the saved session state only when appropriate

await SuspensionManager.RestoreAsync();

}

* + 1. The result of all this is that you get a lot for free. If Contoso Cookbook is suspended and terminated, it automatically goes back to the last page you were viewing when you restart it. You can test this by starting the app from Visual Studio with F5, selecting a recipe, and selecting **Suspend and shutdown** from the **Debug Location** toolbar.
    2. 
    3. After shutting the app down this way, press F5 to restart the app. This simulates what happens when the app is terminated and restarted by the operating system. Thanks to Visual Studio, the app goes back to the recipe you were viewing when you shut it down. Because the recent app history was restored as well, you can even use the back button to retrace your steps through the app.

Task 1 – Modify the OnLaunched method

* 1. App.xaml.cs already contains code to save the navigation state when the app is suspended, and to restore it if it’s terminated. We’ll use a similar strategy to restore the navigation state if the app starts up after being closed by the user, and **Remember where I was** is enabled.
  2. Open App.xaml.cs and add the following using statement near the top.
     1. C#
     2. using Windows.Storage;
  3. Find the **OnLaunched** method. Add the following statements immediately after the **await RecipeDataSource.LoadLocalDataAsync()** clause.
     1. C#
     2. // If the app was closed by the user the last time it ran, and if "Remember
     3. // "where I was" is enabled, restore the navigation state
     4. if (args.PreviousExecutionState == ApplicationExecutionState.ClosedByUser)
     5. {
     6. if (ApplicationData.Current.RoamingSettings.Values.ContainsKey("Remember"))
     7. {
     8. bool remember = (bool)ApplicationData.Current.RoamingSettings.Values["Remember"];
     9. if (remember)
     10. await SuspensionManager.RestoreAsync();
     11. }
     12. }
     13. **Note:** We don’t have to write any code to save the navigation state when the app is closed by the user, because the **Suspending** event fires a few seconds after the app is closed. In fact, there is no event to indicate that an app was closed by the user. When you want to save state before an app shuts down, the **Suspending** event handler is the proper place to do it.

Task 2 – Test the results

* 1. All that remains is to do a little testing to ensure that the change works.
  2. Press F5 to run the app.
  3. Display the charms and select the Settings charm.
  4. Select the **Preferences** command from the settings pane.
  5. Verify that **Remember where I was** is still on. If it’s not, turn it on.
  6. Dismiss the settings pane.
  7. Go to a recipe page.
  8. Close the app by swiping down from the top of the screen or pressing Alt+F4. (*Do not* close the app with Visual Studio’s **Stop Debugging** command.)
  9. Return to Visual Studio and wait a few seconds for the process to end. (It generally takes about 10 seconds.)
  10. Press F5 to start the app again.
  11. Confirm that Contoso Cookbook returns you to the recipe that was displayed when you closed the app.
  12. Go to the preferences page and toggle **Remember where I was** off.
  13. Close the app again by swiping downward from the top of the screen or pressing Alt+F4 while viewing a recipe page.
  14. Return to Visual Studio and wait for the process to end.
  15. Press F5 to start the app again.
  16. Confirm that you go to the app’s start page, and *not* to the recipe you last viewed.
  17. Return to Visual Studio and stop debugging.

Summary

* 1. Settings and preferences are an important part of virtually every Windows Store app. The Settings charm provides a familiar and consistent model for viewing and editing app settings, and as you learned in this lab, it’s simple for an app to expose content through the settings pane. The Callisto library provides a helping hand by providing a **SettingsFlyout** control to host your settings pages, and the pages themselves are easily implemented as user controls.
  2. We’ve come a long way since Lab 1, but there is still more to do. Next up is another important step on the road to Windows Store app stardom: tiles and notifications.