

Hands-on lab

Lab 4: App bars and media capture

September 2012

* 1. 

Contents

[Overview 3](#_Toc334625103)

[Exercise 1: Add an app bar 4](#_Toc334625104)

[Task 1 – Install Callisto 4](#_Toc334625105)

[Task 2 – Add an app bar to the item-detail page 4](#_Toc334625106)

[Task 3 – Test the results 7](#_Toc334625107)

[Exercise 2: Add photo capture 8](#_Toc334625108)

[Task 1 – Use CameraCaptureUI to capture photos 8](#_Toc334625109)

[Task 2 – Enable webcam access 9](#_Toc334625110)

[Task 3 – Test the results 10](#_Toc334625111)

[Exercise 3: Add video capture 10](#_Toc334625112)

[Task 1 – Use CameraCaptureUI to capture video 11](#_Toc334625113)

[Task 2 – Test the results 12](#_Toc334625114)

[Summary 13](#_Toc334625115)

Overview

* 1. The Windows Runtime’s **Windows.Media.Capture** namespace contains a handy set of classes for capturing media and incorporating it into a Windows Store app. In particular, the **CameraCaptureUI** class makes it a breeze to snap photos or capture videos from a webcam. **CameraCaptureUI** even handles devices that include two cameras (one front-facing, the other back-facing), and provides a simple and intuitive UI for switching between the two.
  2. In this lab, you will enhance Contoso Cookbook by allowing users to capture photos and videos of their favorite recipe creations and share them with other apps. You will also add an app bar that provides shortcuts to these features and learn how to incorporate popup menus into app bar commands.

# Objectives

* 1. This lab will show you how to:
  + Implement app bars in a Windows Store app.
  + Add commands and menus to the app bar.
  + Use the Windows Runtime to snap photos.
  + Use the Windows Runtime to capture videos.
  + Share photos and videos by using Share contracts.

# 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 app bar
  3. Add photo capture
  4. Add video capture
  5. Estimated time to complete this lab: **30 to 40 minutes**.

Exercise 1: Add an app bar

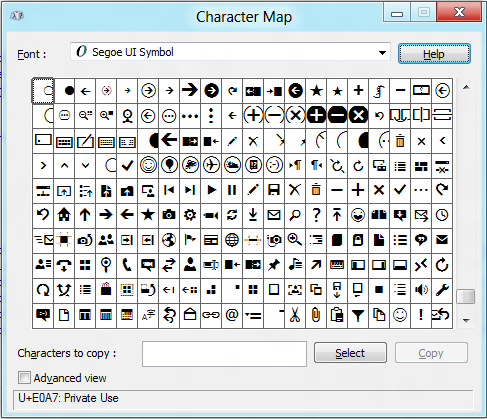
* 1. Before we add features that allow Contoso Cookbook users to capture and share photos and videos, we need to modify the user interface to provide access to those features. An app bar is the perfect tool for the job.

Task 1 – Install Callisto

* 1. Before we add an app bar, we need to install the Callisto control suite. Callisto is a community project led by Tim Heuer that contains helpful controls for Windows Store apps that use XAML, including a Menu control and a Flyout control. We’ll be using these controls in the next task.
  2. Open the ContosoCookbook project you finished in Lab 3 in Visual Studio. If you didn’t complete Lab 3 or would like to start with a reference copy, you’ll find a completed version of the lab in the starting materials.
  3. Go to <http://visualstudiogallery.msdn.microsoft.com/0526563b-7a48-4b17-a087-a35cea701052> and click the **Download** button to download Callisto.vsix.
  4. Run Callisto.vsix and follow the on-screen instructions to install the library.
  5. Return to Visual Studio. In Solution Explorer, right-click **References** and use the **Add Reference** command to add a reference to Callisto to the project. You’ll find it under Windows Extensions.

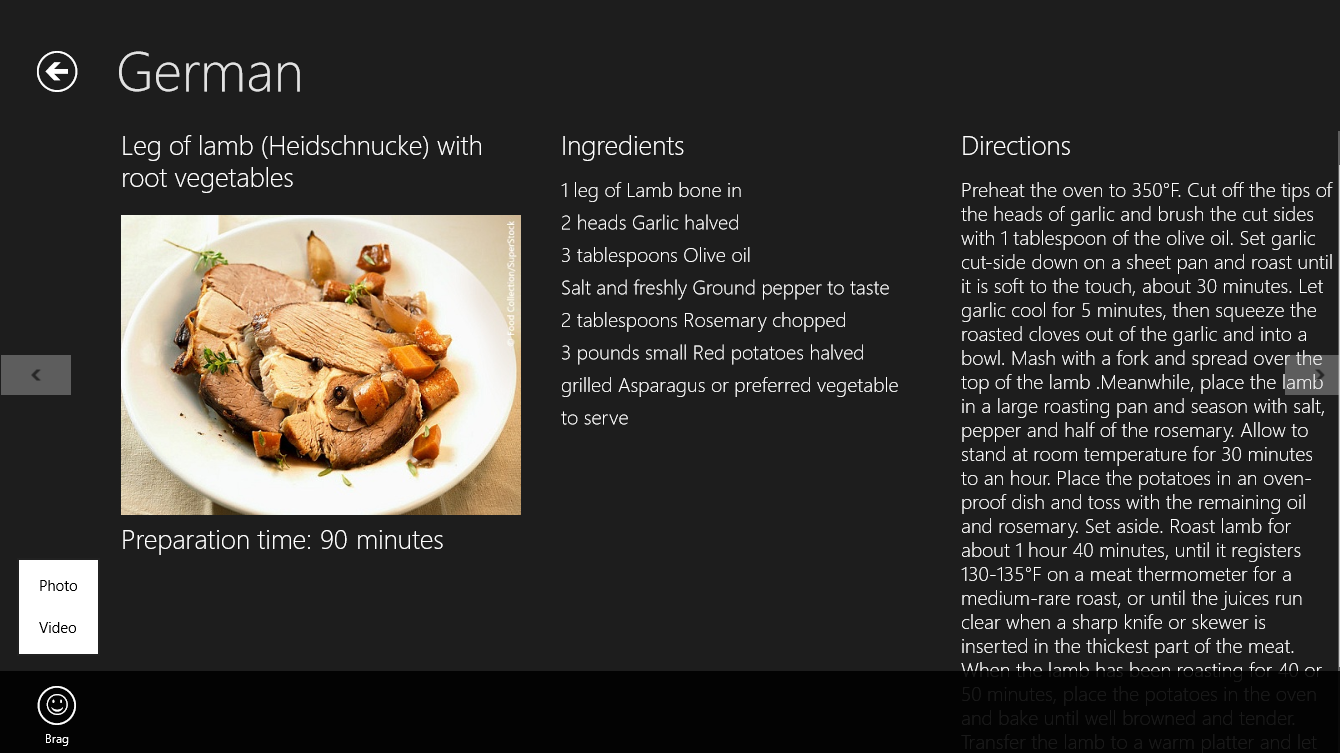
Task 2 – Add an app bar to the item-detail page

* 1. Now we’re ready to add an app bar to the item-detail page. We’ll include a Brag button for capturing photos and videos, and we’ll use a pop-up menu to let the user choose between the two.
  2. Open ItemDetailPage.xaml and add the following statements to the <Page.Resources> section.
     1. XAML
     2. <Style x:Key="BragAppBarButtonStyle" TargetType="Button" BasedOn="{StaticResource AppBarButtonStyle}">
     3. <Setter Property="AutomationProperties.AutomationId" Value="BragAppBarButton"/>
     4. <Setter Property="AutomationProperties.Name" Value="Brag"/>
     5. <Setter Property="Content" Value="&#xE170;"/>
     6. </Style>
  3. Now add the following statements *after* the <Page.Resources> section.
     1. XAML
     2. <Page.BottomAppBar>
     3. <AppBar x:Name="PageAppBar" Padding="10,0,10,0">
     4. <Grid>
     5. <Grid.ColumnDefinitions>
     6. <ColumnDefinition Width="50\*"/>
     7. <ColumnDefinition Width="50\*"/>
     8. </Grid.ColumnDefinitions>
     9. <StackPanel x:Name="LeftCommands" Orientation="Horizontal" Grid.Column="0" HorizontalAlignment="Left">
     10. <Button x:Name="BragButton" HorizontalAlignment="Left" Style="{StaticResource BragAppBarButtonStyle}" Click="OnBragButtonClicked" />
     11. </StackPanel>
     12. <StackPanel x:Name="RightCommands" Orientation="Horizontal" Grid.Column="1" HorizontalAlignment="Right">
     13. </StackPanel>
     14. </Grid>
     15. </AppBar>
     16. </Page.BottomAppBar>
     17. **Note:** The button, or “command,” declared in this app bar gets its look from the BragAppBarButtonStyle that you declared in the previous step. StandardStyles.xaml, which Visual Studio created in the project’s Common folder, contains predefined styles for about 30 different types of app bar commands. If you examine these styles, you’ll find that the symbols on the face of the commands are specified with hexadecimal values such as xE10F. These are character codes in the Segoe UI Symbol character set. To find cool icons for your own commands, fire up the Windows 8 Character Map app, select **Segoe UI Symbol** in the drop-down font list, and scroll to the bottom. Instant iconography!

****

* 1. Open ItemDetailPage.xaml.cs and add the following using statements.
     1. C#
     2. using Callisto.Controls;
     3. using Windows.ApplicationModel.DataTransfer;
  2. Now add the following methods to the ItemDetailPage class.
     1. C#
     2. private void OnBragButtonClicked(object sender, RoutedEventArgs e)
     3. {
     4. // Create a menu containing two items
     5. var menu = new Menu();
     6. var item1 = new MenuItem { Text = "Photo" };
     7. item1.Tapped += OnCapturePhoto;
     8. menu.Items.Add(item1);
     9. var item2 = new MenuItem { Text = "Video" };
     10. item2.Tapped += OnCaptureVideo;
     11. menu.Items.Add(item2);
     12. // Show the menu in a Flyout anchored to the Brag button
     13. var flyout = new Flyout();
     14. flyout.Placement = PlacementMode.Top;
     15. flyout.HorizontalAlignment = HorizontalAlignment.Left;
     16. flyout.HorizontalContentAlignment = HorizontalAlignment.Left;
     17. flyout.PlacementTarget = BragButton;
     18. flyout.Content = menu;
     19. flyout.IsOpen = true;
     20. }
     21. private void OnCapturePhoto (object sender, TappedRoutedEventArgs e)
     22. {
     23. DataTransferManager.ShowShareUI();
     24. }
     25. private void OnCaptureVideo(object sender, TappedRoutedEventArgs e)
     26. {
     27. DataTransferManager.ShowShareUI();
     28. }
     29. **Note:** The **DataTransfer.DataTransferManager.ShowShareUI** method displays the share UI, which is the same UI that appears when you select the Share charm in the charms bar. You’re stubbing out the handlers for the **Photo** and **Video** commands here to show the share UI.

Task 3 – Test the results

* 1. Now let’s see the app bar in action.
  2. Press F5 to run the app.
  3. Tap a recipe to go to the item-detail page.
  4. Display the app bar by swiping upward from the bottom of the screen, right-clicking with the mouse, or pressing Windows logo key+Z.
  5. Verify that the app bar contains a **Brag** command, and that tapping it displays a pop-up menu, as shown in Figure 1.
     1. 
     2. Figure 1
     3. The item-detail page’s app bar
  6. Tap the **Photo** command in the menu and verify that the sharing pane appears.
  7. Return to Visual Studio and stop debugging.

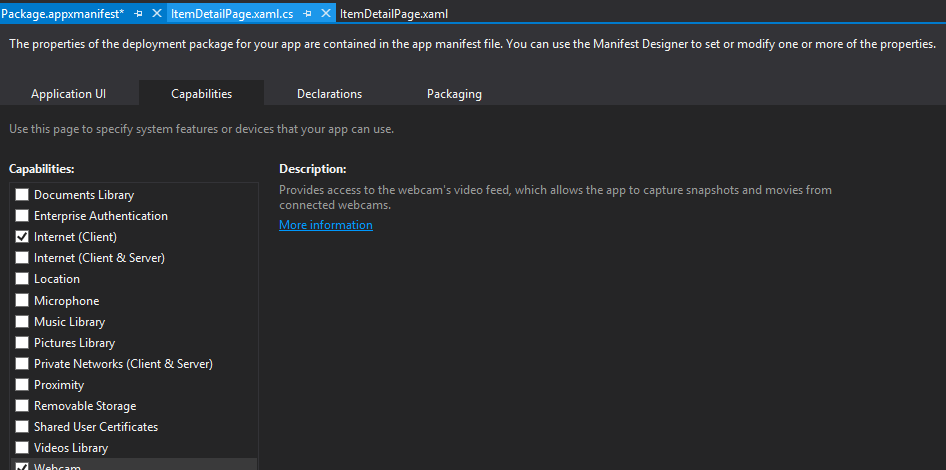
Exercise 2: Add photo capture

* 1. The UI for snapping a photo is in place. Now let’s modify the code to allow the user to take a photo and share it with other apps. The Windows Runtime’s **Windows.Media.Capture** namespace includes a class named **CameraCaptureUI** class that provides a high-level interface to camera hardware. It makes interfacing with cameras about as simple as it could possibly be.

Task 1 – Use CameraCaptureUI to capture photos

* 1. **CameraCaptureUI** has a **CaptureFileAsync** method that makes short work of snapping photos, so let’s put it to work in Contoso Cookbook. While you’re at it, let’s modify the sharing code in the item-detail page so it can share captured photos as well as recipes.
  2. Open ItemDetailPage.xaml.cs and add the following using statements at the top of the file.
     1. C#
     2. using Windows.Media.Capture;
     3. using Windows.Storage;
  3. Add the following field to the ItemDetailPage class.
     1. C#
     2. private StorageFile \_photo; // Photo file to share
  4. Find the OnCapturePhoto method you added in the previous exercise.
  5. Modify the method to look like this. **Be sure to add the async keyword on the first line**. It’s required because we’re using C#’s **await** keyword in the method body.
     1. C#
     2. private async void OnCapturePhoto(object sender, TappedRoutedEventArgs e)
     3. {
     4. var camera = new CameraCaptureUI();
     5. var file = await camera.CaptureFileAsync(CameraCaptureUIMode.Photo);
     6. if (file != null)
     7. {
     8. \_photo = file;
     9. DataTransferManager.ShowShareUI();
     10. }
     11. }
  6. Replace the OnDataRequested method, which you added to ItemDetailPage.xaml.cs in the previous lab, with the one below. These changes are needed to share a recipe if *\_photo* is null, or share a photo if it’s not null.
     1. C#
     2. void OnDataRequested(DataTransferManager sender, DataRequestedEventArgs args)
     3. {
     4. var request = args.Request;
     5. var item = (RecipeDataItem)this.flipView.SelectedItem;
     6. request.Data.Properties.Title = item.Title;
     7. if (\_photo != null)
     8. {
     9. request.Data.Properties.Description = "Recipe photo";
     10. var reference = Windows.Storage.Streams.RandomAccessStreamReference.CreateFromFile(\_photo);
     11. request.Data.Properties.Thumbnail = reference;
     12. request.Data.SetBitmap(reference);
     13. \_photo = null;
     14. }
     15. else
     16. {
     17. request.Data.Properties.Description = "Recipe ingredients and directions";
     18. // Share recipe text
     19. var recipe = "\r\nINGREDIENTS\r\n";
     20. recipe += String.Join("\r\n", item.Ingredients);
     21. recipe += ("\r\n\r\nDIRECTIONS\r\n" + item.Directions);
     22. request.Data.SetText(recipe);
     23. // Share recipe image
     24. var reference = RandomAccessStreamReference.CreateFromUri(new Uri(item.ImagePath.AbsoluteUri));
     25. request.Data.Properties.Thumbnail = reference;
     26. request.Data.SetBitmap(reference);
     27. }
     28. }
  7. Press F5 to run the app and tap a recipe to go to the item-detail page.
  8. Display the app bar and tap the **Brag** button. Then tap **Photo** in the menu. What happens?
  9. Return to Visual Studio and stop debugging.

Task 2 – Enable webcam access

* 1. Windows Store apps must have permission to access webcams. That permission comes through the app manifest, which contains metadata about the app. The next step, then, is to edit Contoso Cookbook’s manifest to indicate that it requires access to webcams.
  2. In Solution Explorer, double-click Package.appxmanifest to open it for editing.
  3. Go to the Capabilities tab and check the Webcam box, as shown in Figure 2.
     1. 
     2. Figure 2
     3. Enabling webcam access in the app manifest

Task 3 – Test the results

* 1. Now that webcam access is enabled, let’s capture a photo.
  2. Press F5 to run the app and tap a recipe to go to the item-detail page.
  3. Display the app bar and tap the **Brag** button. Select **Photo** from the menu and click **Allow** if you are asked whether the app can use your webcam.
  4. When the camera-capture UI appears, tap the screen to snap a photo.
  5. Tap **OK** in the lower-right corner of the screen to accept the photo.
  6. When the share UI appears, select a share target such as Share Target Sample App.
  7. Confirm that the photo you just captured is accepted by the share target.
  8. Return to Visual Studio and stop debugging.

Exercise 3: Add video capture

* 1. Contoso Cookbook users can now snap photos and share them with other apps. In this exercise, you’ll add support for capturing videos, too. You’ll change the parameter passed to **CaptureFileAsync** to indicate that you want to capture video rather than photos, and use **CameraCaptureUI**’s **VideoSettings** property to indicate what format you want to capture in.

Task 1 – Use CameraCaptureUI to capture video

* 1. You can use the same **CaptureFileAsync** method that captures a photo to capture video, too. Let’s modify the **OnCaptureVideo** method you stubbed out earlier to demonstrate.
  2. Open ItemDetailPage.xaml.cs and add the following field to the ItemDetailPage class.
     1. C#
     2. private StorageFile \_video; // Video file to share
  3. Find the OnCaptureVideo method you added to ItemDetailPage.xaml.cs in Exercise 1.
  4. Modify the method to look like this. Once more, be sure to include the **async** keyword on the first line. It’s required because we’re using C#’s **await** keyword in the method body.
     1. C#
     2. private async void OnCaptureVideo(object sender, TappedRoutedEventArgs e)
     3. {
     4. var camera = new CameraCaptureUI();
     5. camera.VideoSettings.Format = CameraCaptureUIVideoFormat.Wmv;
     6. var file = await camera.CaptureFileAsync(CameraCaptureUIMode.Video);
     7. if (file != null)
     8. {
     9. \_video = file;
     10. DataTransferManager.ShowShareUI();
     11. }
     12. }
  5. Add an **else-if** clause to the OnDataRequested method so the app can share videos in addition to photos.
     1. C#
     2. void OnDataRequested(DataTransferManager sender, DataRequestedEventArgs args)
     3. {
     4. var request = args.Request;
     5. var item = (RecipeDataItem)this.flipView.SelectedItem;
     6. request.Data.Properties.Title = item.Title;
     7. if (\_photo != null)
     8. {
     9. request.Data.Properties.Description = "Recipe photo";
     10. var reference = Windows.Storage.Streams.RandomAccessStreamReference.CreateFromFile(\_photo);
     11. request.Data.Properties.Thumbnail = reference;
     12. request.Data.SetBitmap(reference);
     13. \_photo = null;
     14. }
     15. else if (\_video != null)
     16. {
     17. request.Data.Properties.Description = "Recipe video";
     18. List<StorageFile> items = new List<StorageFile>();
     19. items.Add(\_video);
     20. request.Data.SetStorageItems(items);
     21. \_video = null;
     22. }
     23. else
     24. {
     25. request.Data.Properties.Description = "Recipe ingredients and directions";
     26. // Share recipe text
     27. var recipe = "\r\nINGREDIENTS\r\n";
     28. recipe += String.Join("\r\n", item.Ingredients);
     29. recipe += ("\r\n\r\nDIRECTIONS\r\n" + item.Directions);
     30. request.Data.SetText(recipe);
     31. // Share recipe image
     32. var reference = RandomAccessStreamReference.CreateFromUri(new Uri(item.ImagePath.AbsoluteUri));
     33. request.Data.Properties.Thumbnail = reference;
     34. request.Data.SetBitmap(reference);
     35. }
     36. }
  6. Go to the Capabilities section of the app manifest and select the **Microphone** box. This is necessary because when you capture video, **CameraCaptureUI** uses the microphone along with the camera.

Task 2 – Test the results

* 1. Now let’s test the code you just added.
  2. Press F5 to run the app and tap a recipe to go to the item-detail page.
  3. Display the app bar, tap the **Brag** button, and select **Video** from the menu.
  4. If asked whether the app can use your camera and microphone, click **Allow**.
  5. When the camera-capture UI appears, tap the screen to begin capturing video.
  6. After a few seconds, tap the screen again to stop capturing video.
  7. Tap **OK** in the lower-right corner of the screen to accept the video.
  8. When the share UI appears, select a share target such as Share Target Sample App.
  9. Confirm that the video you just captured is accepted by the share target.
  10. Return to Visual Studio and stop debugging.

Summary

* 1. On some platforms, it’s difficult to include photo and video capture capabilities in an app because there is a requirement to interface with cameras at the device level. The Windows Runtime makes media capture extraordinarily easy by providing the core UI and logic in the **CameraCaptureUI** class. In Contoso Cookbook, we don’t do anything with captured image and video files other than share them out. However, you could easily use types in the **Windows.Storage** namespace to save these files to the file system and allow the user to create libraries of recipe photos and videos.
  2. You may not have noticed, but the operating system did something pretty cool for you when you checked the **Webcam** and **Microphone** boxes in the manifest. If you run the app, then select the Settings charm and select **Permissions**, you will see that the permissions page now contains a toggle switch for turning camera and microphone access on and off. Users can disable apps that use cameras and microphones, and you don’t have to write a single line of code to make that happen.