Hands-On Lab

Searching and Sharing

Lab version: 1.0.0

Last updated: 6/8/2012



Contents

[Overview 3](#_Toc320016049)

[Exercise 1: Add Recipe Sharing 4](#_Toc320016050)

[Task 1 – Invoke the Share Charm 4](#_Toc320016051)

[Task 2 – Implement Recipe Sharing 4](#_Toc320016052)

[Task 3 – Implement Recipe Image Sharing 7](#_Toc320016053)

[Exercise 2: Add Recipe Search 9](#_Toc320016054)

[Task 1 – Invoke the Search Charm 9](#_Toc320016055)

[Task 2 – Add Search Support 10](#_Toc320016056)

[Task 3 – Add Search Suggestions 13](#_Toc320016057)

[Task 4 – Add Support for External Launching 14](#_Toc320016058)

[Summary 16](#_Toc320016059)

Overview

* 1. One of the key features of the Metro user interface is the charms bar that slides in from the right side of the screen in response to swipes or presses of Win-C. The buttons (“charms”) in the charms bar provide a means for Metro-style apps to expose commonly used features that is consistent across applications. For example, if you want to perform a search in a Metro application, you select the Search charm and type a search term into Metro’s search pane. The UI – and the actions required to invoke that UI – are the same in every application. To share data with another application, you use the Share charm. The app then makes the data that the user is currently viewing – for example, a drawing in a paint application or a recipe in Contoso Cookbook – available to other applications via a sharing contract that is functionally similar to clipboard sharing in traditional Windows desktop applications.
  2. In this lab, you’ll add support for searching and sharing to Contoso Cookbook. You’ll get first-hand experience implementing searching and sharing contracts, and you’ll see how these contracts provide a higher level of integration between a Metro-style app and Metro itself.

# Objectives

* 1. This lab will show you how to:
  + Implement sharing in a Metro-style app
  + Implement search in a Metro-style app
  + Implement search suggestions

# 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 2012 RC for Windows 8

# Exercises

* 1. This Hands-On Lab comprises the following exercises:
  2. Add Recipe Sharing
  3. Add Recipe Search
  4. Estimated time to complete this lab: **30 to 40 minutes**.

Exercise 1: Add Recipe Sharing

1. In Exercise 1, you’ll add sharing support to Contoso Cookbook so recipes can be shared with other applications. You’ll share two types of data for each recipe: textual data that includes the recipe name, ingredients, and directions, and image data containing a pictorial representation of the recipe.

Task 1 – Invoke the Share Charm

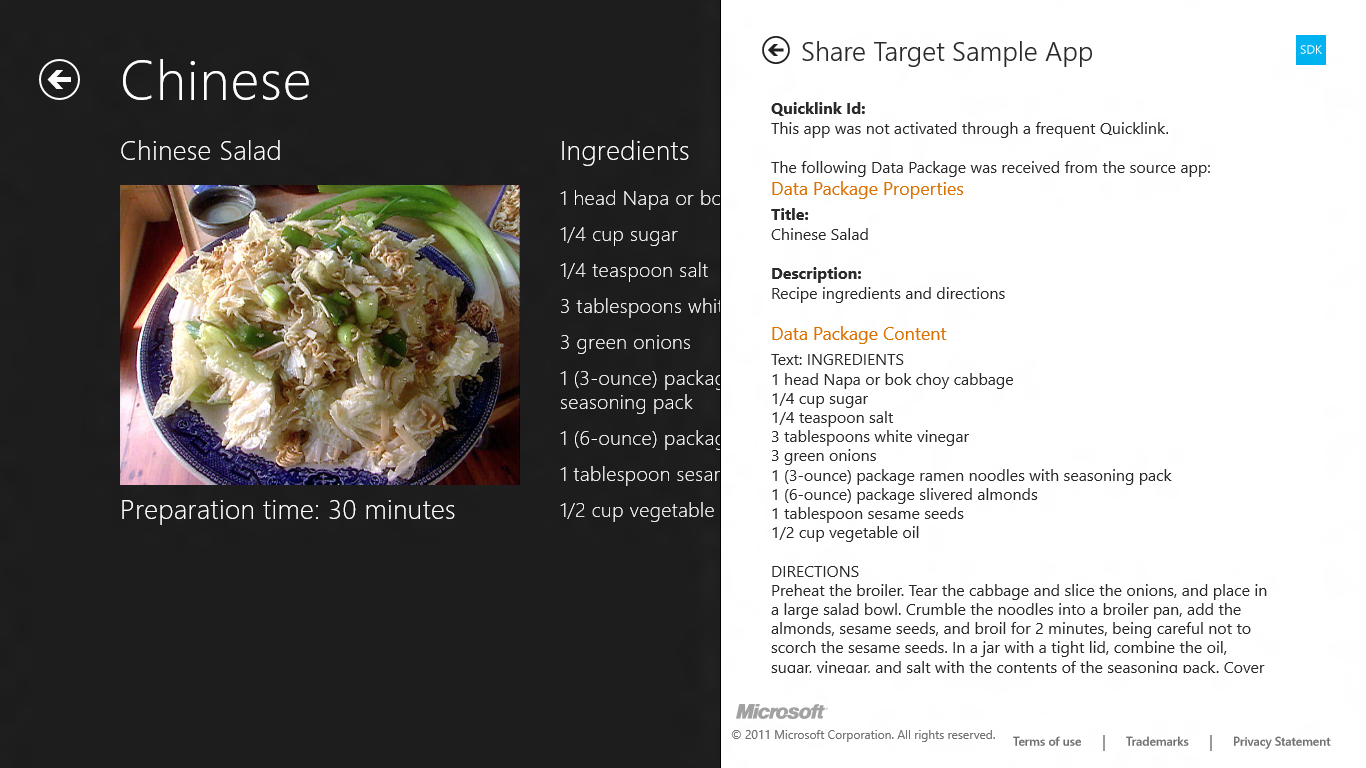
* 1. To get started, let’s see how the Share charm behaves when it’s invoked in Contoso Cookbook before sharing support is added.
  2. Open the ContosoCookbook project you finished in Lab 2 in Visual Studio. If you didn’t complete Lab 2 or would like to start with a reference copy, you’ll find a completed version of the lab in the starting materials.
  3. Press F5 to launch Contoso Cookbook.
  4. Tap one of the recipes to show the item-detail page.
  5. Display the charms bar by swiping from right to left from the right edge of the screen if you’re using a touch screen, or by pressing Win-C if you’re not.
  6. Tap the Share charm to display Metro’s sharing pane.
  7. Since Contoso Cookbook currently doesn’t implement a sharing contract, the sharing pane informs you “Contoso Cookbook can’t share.”
  8. Return to Visual Studio and stop debugging.

Task 2 – Implement Recipe Sharing

Now that you’ve seen what the sharing pane looks like when an app doesn’t support sharing, let’s add a sharing contract to Contoso Cookbook so it can share recipe data. First, we’ll need to add a bit of infrastructure to support that contract.

* 1. Open ItemDetailPage.xaml.cs and add the following using statements at the top of the file:
     1. C#
     2. using Windows.ApplicationModel.DataTransfer;
     3. using System.Text;
     4. using Windows.Storage.Streams;
  2. Add the following fields to the ItemDetailPage class:
     1. C#
     2. private static TypedEventHandler<DataTransferManager, DataRequestedEventArgs> \_handler;
     3. private RecipeDataItem \_item; // Recipe currently displayed
  3. Find the LoadState method and add the highlighted statements:
     1. C#
     2. var item = (RecipeDataItem)e.Parameter;
     3. this.DefaultViewModel["Group"] = item.Group;
     4. this.DefaultViewModel["Items"] = item.Group.Items;
     5. this.flipView.SelectedItem = item;
     6. this.\_item = item;
     7. // Register handler for DataRequested events for sharing
     8. if (\_handler != null)
     9. DataTransferManager.GetForCurrentView().DataRequested -= \_handler;
     11. \_handler = new TypedEventHandler<DataTransferManager, DataRequestedEventArgs>(OnDataRequested);
     12. DataTransferManager.GetForCurrentView().DataRequested += \_handler;
  4. Now add the following method to the ItemDetailPage class:
     1. C#
     2. void OnDataRequested(DataTransferManager sender, DataRequestedEventArgs args)
     3. {
     4. var request = args.Request;
     5. request.Data.Properties.Title = \_item.Title;
     6. request.Data.Properties.Description = "Recipe ingredients and directions";
     7. // Share recipe text
     8. StringBuilder builder = new StringBuilder();
     9. builder.Append("INGREDIENTS\r\n");
     10. foreach (string ingredient in \_item.Ingredients)
     11. {
     12. builder.Append(ingredient);
     13. builder.Append("\r\n");
     14. }
     15. builder.Append("\r\nDIRECTIONS\r\n");
     16. builder.Append(\_item.Directions);
     17. request.Data.SetText(builder.ToString());
     18. }
     19. **Note:** Sharing is implementing by registering a handler for DataTransferManager’s DataRequested events, which fire when the user activates the Share charm. In this example, you’re responding to that event by calling SetText on the DataPackage object exposed through args.Request.Data to provide the recipe in the form of text. When the sharing pane appears with a list of share targets, the list will include only share targets that can consume text.

DataTransferManager only allows one DataRequested event handler to be registered at any given time. To be certain that the previous event handler is deregistered, even if it belongs to another instance of the ItemDetailPage class, we store a reference to it in a static field and deregister it before attempting to register another one.

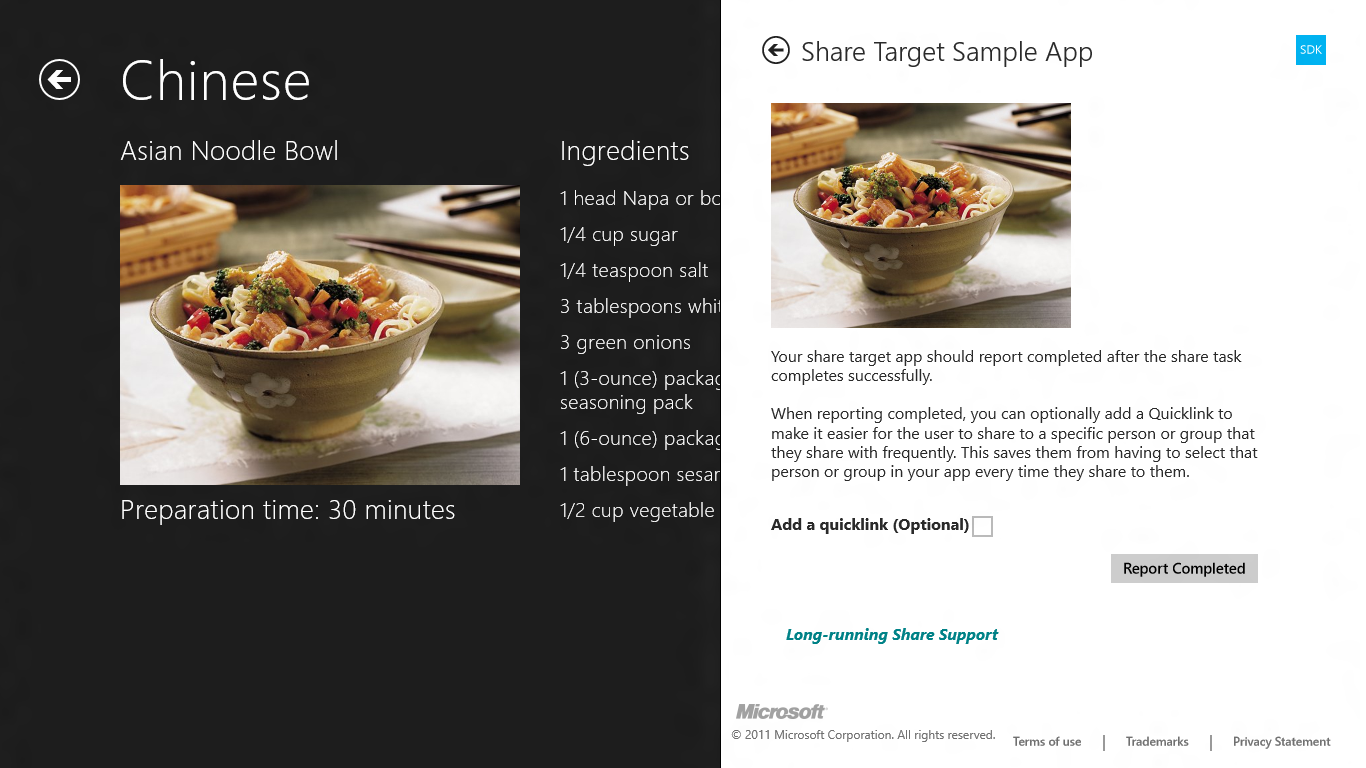
* 1. Open ItemDetailPage.xaml and add a SelectionChanged=”FlipView\_SelectionChanged” attribute to all three FlipView controls:
     1. XAML
     2. <FlipView
     3. x:Name="flipView"
     4. SelectionChanged="FlipView\_SelectionChanged"
     5. .
     6. .
     7. .
     8. <FlipView
     9. x:Name="portraitFlipView"
     10. SelectionChanged="FlipView\_SelectionChanged"
     11. .
     12. .
     13. .
     14. <FlipView
     15. x:Name="SnappedFlipView"
     16. SelectionChanged="FlipView\_SelectionChanged"
     17. .
     18. .
     19. .
  2. Go back to ItemDetailPage.xaml.cs and add the following method to the ItemDetailPage class:
     1. C#
     2. private void FlipView\_SelectionChanged(object sender, SelectionChangedEventArgs e)
     3. {
     4. this.\_item = (sender as FlipView).SelectedItem as RecipeDataItem;
     5. }
     6. **Note:** The purpose of the event handler you just added is to make sure that \_item, which stores a reference to the recipe that’s currently displayed, is updated with a reference to the current recipe if, after navigating to the item-detail page, the user scrolls horizontally to another recipe.
  3. Press F5 to launch the application.
  4. Tap one of the recipes to show the recipe-detail page.
  5. Display the charms bar and select the Share charm to display Metro’s sharing pane. The sharing pane now shows you a list of *share targets* installed on your device – applications that can consume data shared by share sources.
     1. **Note:** If you haven’t done so already, now would be a great time to install the share-target sample that comes with the Windows 8 SDK. Called Share Target Sample App, it demonstrates how to write share-target applications. More importantly, it provides a sharing target to test with as you develop applications that act as sharing sources, and it accepts images as well as text and other data types. To install the share-target sample, open it in Visual Studio and run it one time. After that, it should appear in the list of share targets when you select the Share charm from a Metro application.
  6. Select one of the share targets listed in the sharing pane and verify that it received the recipe data. Figure 1 shows how the Windows 8 SDK’s Share Target Sample App looks after accepting recipe text shared by Contoso Cookbook.
  7. 
  8. Figure 1
  9. Share Target Sample App showing a shared recipe
  10. Return to Visual Studio and stop debugging.

Task 3 – Implement Recipe Image Sharing

Contoso Cookbook can now share textual recipe data, but since each recipe is accompanied by an image, it should share recipe images as well. That way, a share target that accepts images can show a photo of the recipe along with the recipe text (assuming the share target supports text as well as images). Let’s revise your sharing code to support bitmaps as well as text.

* 1. Return to the OnDataRequested method you added in the previous task.
  2. Add the following statements to the end of the method:
     1. C#
     2. // Share recipe image
     3. string url = \_item.GetImageUri();
     4. if (!url.StartsWith("http://"))
     5. url = "ms-appx:///" + url;
     6. var uri = new Uri(url);
     7. var reference = RandomAccessStreamReference.CreateFromUri(uri);
     8. request.Data.Properties.Thumbnail = reference;
     9. request.Data.SetBitmap(reference);

**Note:** To share an image from a page, you pass the image URI to RandomAccessStreamReference.CreateFromUri, and then pass the returned RandomAccessStreamReference to DataPackage.SetBitmap. CreateFromUri is supposed to work with local as well as remote images, but in the Windows 8 Consumer Preview, it doesn’t. The if clause in the code you just added is a work-around that shouldn’t be necessary in the final version of Windows 8.

* 1. Press F5 to launch the application.
  2. Tap one of the recipes to show the recipe-detail page.
  3. Display the charms bar and select the Share charm to display Metro’s sharing pane.
  4. Select one of the share targets listed in the sharing pane and verify that it received the recipe image. Figure 2 shows how the Share Target Sample App looks after accepting a recipe image from Contoso Cookbook.
  5. 
  6. Figure 2
  7. Share Target Sample App showing a shared recipe image
  8. Return to Visual Studio and stop debugging.

Exercise 2: Add Recipe Search

1. In Exercise 2, you’ll add search support to Contoso Cookbook so users can use the Search charm to search for recipe data. A user who wants to find all recipes that contain sugar, for example, will invoke the Search charm, type “sugar” into the search box, and be presented with a list of sugary recipes.

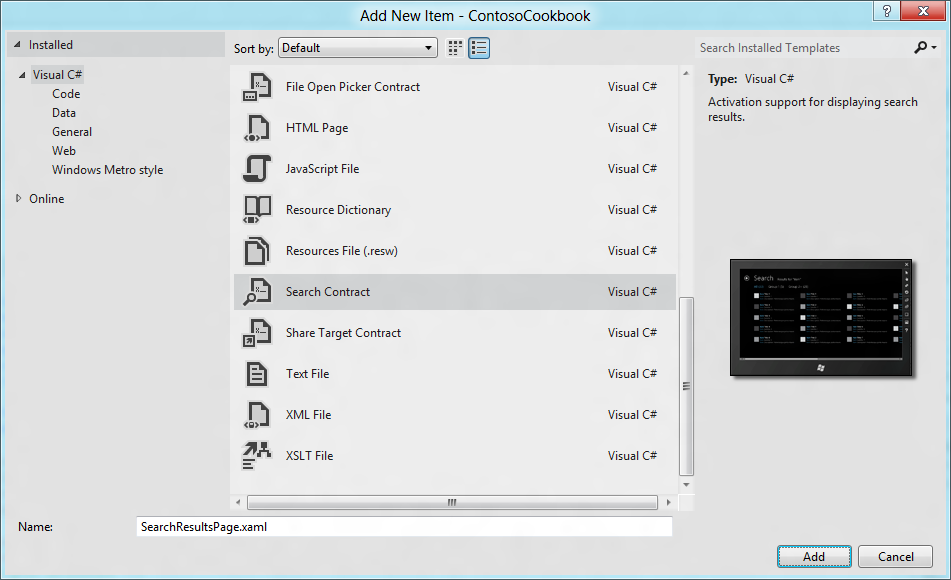
Task 1 – Invoke the Search Charm

Before adding search support to Contoso Cookbook, let’s see what Metro’s search UI looks like when it’s invoked while Contoso Cookbook is the foreground application.

* 1. Press F5 to launch the application.
  2. Display the charms bar by swiping from right to left from the right edge of the screen if you’re using a touch screen, or by pressing Win-C if you’re not.
  3. Tap the Search charm to display Metro’s search pane.
  4. Type “sugar” (without the quotation marks) into the search box and press Enter or tap the magnifying-glass icon at the right end of the search box.
  5. Windows 8 tells you “No apps match your search.” That will change once you add search support.
  6. Return to Visual Studio and stop debugging.

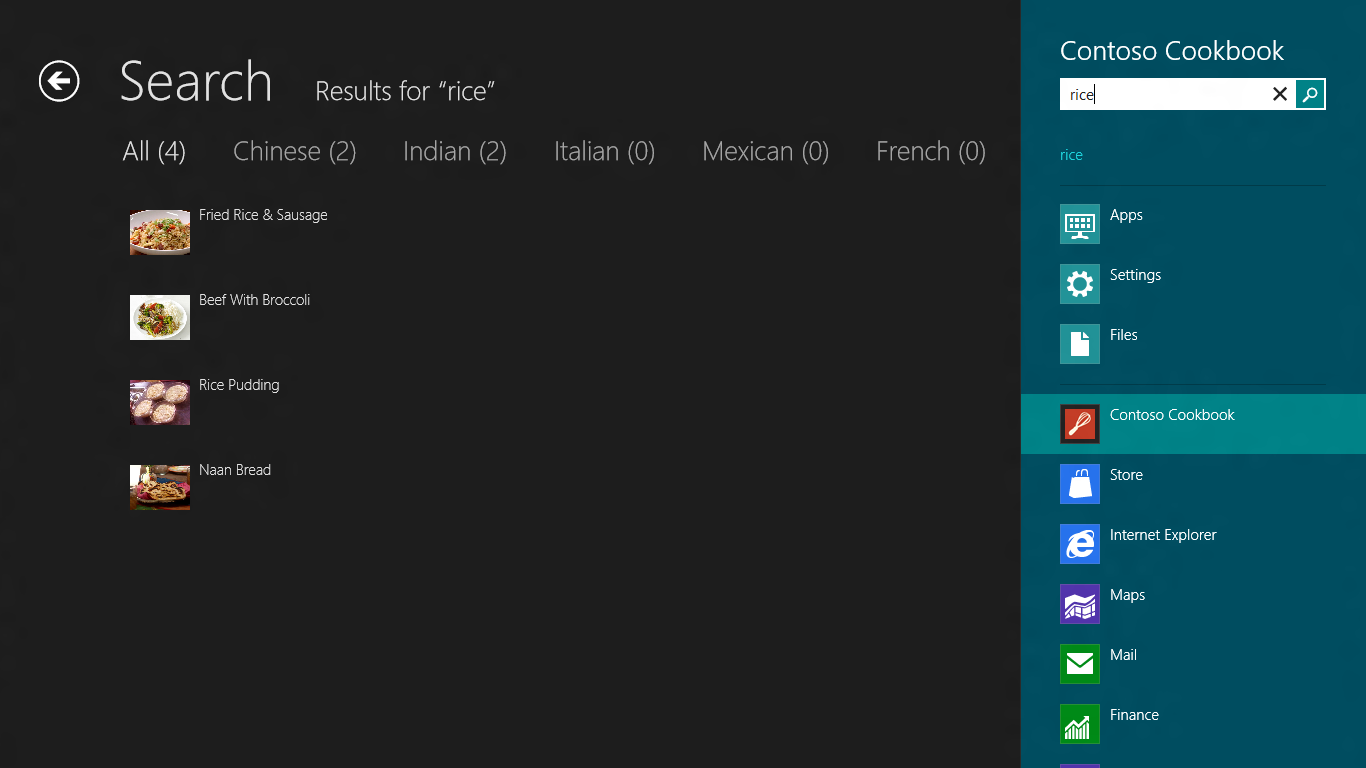
Task 2 – Add Search Support

To implement search, you code a search contract for your application. Visual Studio will do most of the work for you by inserting a C# contract into your app. You’ll need to tweak the code to perform domain-specific searches within your application’s data. It’s easy to do, as the next few steps will demonstrate.

* 1. Right-click the project in Solution Explorer and use the **Add - New Item** command to add a search contract named SearchResultsPage.xaml, as shown in Figure 3.
  2. 
  3. Figure 3
  4. Adding a search contract
  5. In the <Page.Resources> section of SearchResultsPage.xaml, change “My Application” to “Search” in the string resource named “AppName” to change the title at the top of the page:
     1. XAML
     2. <x:String x:Key="AppName">Search</x:String>
  6. Add the following attributes to the GridView named “resultsGridView” further down in the file:
     1. XAML
     2. ItemClick="OnItemClick"
  7. Add the same attributes to the ListView named “resultsListView” a little further down. This ListView is only displayed if search results are displayed while Contoso Cookbook is snapped.
  8. Open StandardStyles.xaml in the Common folder.
  9. Find the DataTemplate resource named “StandardSmallIcon300x70ItemTemplate.” Remove the TextBlock bound to the Subtitle property and apply the highlighted changes:
     1. XAML
     2. <DataTemplate x:Key="StandardSmallIcon300x70ItemTemplate">
     3. <Grid Width="300">
     4. <Grid.ColumnDefinitions>
     5. <ColumnDefinition Width="Auto"/>
     6. <ColumnDefinition Width="\*"/>
     7. </Grid.ColumnDefinitions>
     8. <Border Background="{StaticResource ListViewItemPlaceholderRectBrush}" Margin="10,10,0,20" Width="60" Height="45">
     9. <Image Source="{Binding Image}" Stretch="UniformToFill"/>
     10. </Border>
     11. <StackPanel Grid.Column="1" Margin="10,0,10,10">
     12. <TextBlock Text="{Binding ShortTitle}" Style="{StaticResource BodyTextStyle}" TextWrapping="NoWrap"/>
     13. </StackPanel>
     14. </Grid>
     15. </DataTemplate>
  10. Open SearchResultsPage.xaml.cs and add the following using statement at the top of the file:
      1. C#
      2. using ContosoCookbook.Data;
  11. Next, add the following field to the SearchResultsPage class:
      1. C#
      2. // Collection of RecipeDataItem collections representing search results
      3. private Dictionary<string, List<RecipeDataItem>> \_results = new Dictionary<string, List<RecipeDataItem>>();
  12. Go to the LoadState method and find the following statements:
      1. C#
      2. this.DefaultViewModel["Filters"] = filterList;
      3. this.DefaultViewModel["ShowFilters"] = filterList.Count > 1;
  13. Add the following statements *before* the two statements you identified in the previous step:
      1. C#
      2. // Search recipes and tabulate results
      3. string query = queryText.ToLower();
      4. var all = new List<RecipeDataItem>();
      5. \_results.Add("All", all);
      6. foreach (var group in RecipeDataSource.GetGroups("AllGroups"))
      7. {
      8. var items = new List<RecipeDataItem>();
      9. \_results.Add(group.Title, items);
      11. foreach (var item in group.Items)
      12. {
      13. if (item.Title.ToLower().Contains(query) || item.Directions.ToLower().Contains(query))
      14. {
      15. all.Add(item);
      16. items.Add(item);
      17. }
      18. }
      19. filterList.Add(new Filter(group.Title, items.Count, false));
      20. }
      21. filterList[0].Count = all.Count;

**Note:** The code you just added searches all recipe titles and directions for the query text entered by the user. For each matching recipe it finds, it adds a RecipeDataItem to a List representing all recipes, and a RecipeDataItem to a List representing recipes in a specific group. The Lists are maintained in the Dictionary you declared a moment ago and are tagged with group names such as “All,” “Chinese,” and so on.

This code also populates the filter list (represented by the filterList variable) at the top of the search results page with group names showing the number of matching recipes in each group.

* 1. Find the Filter\_SelectionChanged method and add the following statement to the end of the if clause (right after the TODO comment):
     1. C#
     2. this.DefaultViewModel["Results"] = \_results[selectedFilter.Name];
  2. Add the following method to the SearchResultsPage class to navigate to the recipe page when an item is clicked in the search results page:
     1. C#
     2. private void OnItemClick(object sender, ItemClickEventArgs e)
     3. {
     4. // Navigate to the page showing the recipe that was clicked
     5. this.Frame.Navigate(typeof(ItemDetailPage), ((RecipeDataItem)e.ClickedItem).UniqueId.ToString());
     6. }
  3. Press F5 to start the application.
  4. Display the charms bar.
  5. Tap the Search charm to display Metro’s search pane.
  6. Type “rice” (without the quotation marks) into the search box at the top of the search pane and press Enter or tap the magnifying-glass icon at the right end of the search box.
  7. Verify that four recipes appear in the search results (see Figure 4).
  8. 
  9. Figure 4
  10. Search results for “rice”
  11. Select one of the recipes and verify that the recipe detail appears.
  12. Return to Visual Studio and stop debugging.

Task 3 – Add Search Suggestions

A useful enhancement that you can add to the search experience is to provide suggestions as the user types a search term into the search box. It’s easy to do; all you have to do is add a handler for SuggestionsRequested events. Here’s how.

* 1. Open App.xaml.cs and add the following using statement at the top of the file:
     1. C#
     2. using Windows.ApplicationModel.Search;
  2. Add the following statements to the end of the OnLaunched method:
     1. C#
     2. // Register handler for SuggestionsRequested events from the search pane
     3. SearchPane.GetForCurrentView().SuggestionsRequested += OnSuggestionsRequested;
  3. Now add the following method to the App class:
     1. C#
     2. void OnSuggestionsRequested(SearchPane sender, SearchPaneSuggestionsRequestedEventArgs args)
     3. {
     4. string query = args.QueryText.ToLower();
     5. string[] terms = { "salt", "pepper", "water", "egg", "vinegar", "flour", "rice", "oil" };
     7. foreach(var term in terms)
     8. {
     9. if (term.StartsWith(query))
     10. args.Request.SearchSuggestionCollection.AppendQuerySuggestion(term);
     11. }
     12. }
     13. **Note:** The code you just added provides search suggestions for words that pattern-match salt, pepper, water, egg, vinegar, flour, rice, and oil. If the user types “sa,” the word “salt” will appear in the search pane as a suggested completion term. Of course, you can add as many suggestions as you’d like. If you want “ketchup” to appear when the user types “ke,” simply add that term to the list.
  4. Start the application again and type “vi” into the search box. Verify that “vinegar” appears in the suggestion list underneath the search box, as shown in Figure 5.
  5. 
  6. Figure 5
  7. Search suggestions in action
  8. Return to Visual Studio and stop debugging.

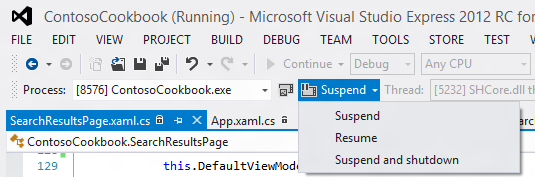
Task 4 – Add Support for External Launching

We’re almost done, but there is one task left to perform. Currently, search works great if Contoso Cookbook is running when a search is executed. But it doesn’t work at all if the user invokes Search from outside the application. To see for yourself, make sure Contoso Cookbook isn’t running (you can check for it in Task Manager and kill it if needed). Then invoke the Search charm, type a search term, and tap Contoso Cookbook in the list of applications in the search pane. In a moment, you’ll find yourself staring at a blank screen. Let’s fix that.

* 1. Open App.xaml.cs and find the OnSearchActivated method at the bottom.
     1. **Note:** OnSearchActivated is a key virtual method in the Application class. It’s called to activate the search-results page when the user invokes Search with the application running. It’s also called in lieu of OnLaunched if the user selects Contoso Cookbook in the search pane. If the latter occurs, we need to load the recipe data and activate the search-results page after the asynchronous load completes.
  2. following using statements at the top of the file
     1. C#
     2. using ContosoCookbook.Data;
  3. OnSearchActivated 메서드를 찾아서 아래 두줄을 추가해준다. 이건 검색을 통해서 앱이 Activate 되었을 때 데이터를 로드하는 역할을 한다. await 키워드를 썼으니 메서드 선언부에 async를 잊지 않는다.
     1. C#
     2. protected async override void OnSearchActivated(SearchActivatedEventArgs args)
     3. {
     4. await RecipeDataSource.LoadRemoteDataAsync();
     5. SearchPane.GetForCurrentView().SuggestionsRequested += OnSuggestionsRequested;
     6. ContosoCookbook.SearchResultsPage.Activate(args.QueryText, args.PreviousExecutionState);
     7. }
  4. SearchResultsPage.xaml.cs 를 열어서Activate 메서드에서 else 구문을 찾는다. 아래처럼 기존 코드는 지우고 새로운 코드를 넣어준다. 이건 앱이 terminated 된 상태에서 검색을 통해 Activate 된 상황처럼 앱의 기본 골격인 frame 컨트롤이 없는 경우 새로 만들어 주는 것이다.
     1. C#
     2. else
     3. {
     4. ~~// Otherwise bypass navigation and provide the tools needed to emulate the back stack~~
     5. ~~SearchResultsPage page = new SearchResultsPage();~~
     6. ~~page.\_previousContent = previousContent;~~
     7. ~~page.\_previousExecutionState = previousExecutionState;~~
     8. ~~.LoadState(queryText, null);~~
     9. ~~Window.Current.Content = page;~~
     10. frame = new Frame();
     11. Window.Current.Content = frame;
     12. ~~frame.Navigate(typeof(SearchResultsPage), new Tuple<String, UIElement>(queryText, previousContent));~~
     13. frame.Navigate(typeof(SearchResultsPage), queryText);
     14. }
  5. GoBack 메서드의else 구문을 찾아 아래 코드를 넣어준다. (TODO 커멘트 바로 아래) : 이건 앱이 Terminated 된 상태에서 검색을 통해 Active 된 경우처럼 검색 결과 페이지의 뒤로 버튼을 누르면 어디로 갈지를 명확히 해준다. 여기서는 앱의 첫페이지로 돌아갔다.

다시 설명하면 앱이 첫페이지부터 표시하면서 정상적으로 Active 되지 않은 상황이라면 검색 결과 페이지에서 뒤로 돌아갈 곳이 없으므로 만들어 주는 것이다.

* + 1. C#
    2. else
    3. {
    4. // TODO: invoke the app's normal launch behavior, using this.\_previousExecutionState
    5. // as appropriate. Exact details can vary from app to app, which is why an
    6. // implementation isn't included in the Search Contract template. Typically
    7. // this method and OnLaunched in App.xaml.cs can call a common method.
    8. this.Frame.Navigate(typeof(GroupedItemsPage), "AllGroups");
    9. }
    10. **Note:** These statements disable the search-results page’s back button when the app is launched to perform a search. Without these statements, a click of the back button goes to an empty page.
  1. F5를 눌러서 앱을 실행시킨 후 다시 Visual Studio로 돌아온다. 앱을 완전히 Terminate 시켜야 테스트가 가능하므로 디버깅 상태의 Visual Studio 에서 툴바에서 Suspend and Shutdown 버튼을 눌러 앱을 완전히 Terminate 시킨 후



* 1. Use Task Manager to make sure Contoso Cookbook isn’t running. End the process if it is.
  2. Display the charms bar and tap the Search charm.
  3. Enter a search term such as “rice” or “sugar.” Then tap Contoso Cookbook in the list of applications in the search pane.
  4. Verify that Contoso Cookbook starts up and displays the correct search results,
  5. Back 버튼을 눌러서 가장 상위 페이지로 이동하는지 테스트 한다.
  6. Press Alt-F4 to close the application.

Summary

* 1. Contracts are an important part of Windows 8 because they allow apps to integrate with the Metro shell and provide a user experience that is both consistent and predictable. This loosely coupled collaboration is very synergistic and extensible, allowing you to share anything with any app, search within any app, and so on.
  2. In this lab, you saw two types of contracts first-hand: sharing contracts and search contracts. In a later lab, you’ll use another type of contract to integrate with the Settings charm. But first there’s something else we need to tackle: media capture using the cameras built into most PCs and mobile devices today. That’s the subject of the next lab, so let’s get started!