

App Clinic Lab Manual

The Contoso Electronics App

App Clinic Lab Manual

The Field Engineer App

This lab provides an introduction to the main design and functional aspects of a Windows Store-type retail product catalog app. It shows participants how to create a simple product hub using tiles, tied to details pages with associated product information. Attendees begin with a new project and go about the task of building the various pieces of the application. The solution uses code snippets to enable the rapid development of the complete application within the scope of a 5-6 hour, instructor-led session.

Windows 8.1 App Clinic

9.17.2013 | Version 1.0

This document is provided “as-is.” Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Copyright © 2013 Microsoft Corporation. All rights reserved.

Microsoft Windows and Visual Studio are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Contents

[Introduction 1](#_Toc367109544)

[Overview 2](#_Toc367109545)

[Welcome to the App Clinic Contoso Electronics Lab 2](#_Toc367109546)

[Objectives 2](#_Toc367109547)

[Lab Structure 2](#_Toc367109548)

[Estimated Time for Completion 3](#_Toc367109549)

[Getting Started 4](#_Toc367109550)

[System Requirements 5](#_Toc367109551)

[Hardware 5](#_Toc367109552)

[Software 5](#_Toc367109553)

[Solution Overview 6](#_Toc367109554)

[Solution Structure 6](#_Toc367109555)

[Navigation Structure 7](#_Toc367109556)

[Getting to Know the Application 8](#_Toc367109557)

[Branding Considerations 10](#_Toc367109558)

[Understanding Visual Elements 10](#_Toc367109559)

[Image Assets/Logos 10](#_Toc367109560)

[Understanding XAML Styling 11](#_Toc367109561)

[Data Considerations 13](#_Toc367109562)

[Data Model 13](#_Toc367109563)

[Data Schema 14](#_Toc367109564)

[Lab Exercises 16](#_Toc367109565)

[Exercise 1: Local Data Binding 17](#_Toc367109566)

[Task 1: Set Up the Data XML files 17](#_Toc367109567)

[Task 2: Define the Data Source 17](#_Toc367109568)

[Task 3: Fetch and Bind the Data 19](#_Toc367109569)

[Exercise 2: Branding 20](#_Toc367109570)

[Task 1: Configure Visual Assets 20](#_Toc367109571)

[Task 2: Customize Background & Foreground Colors 21](#_Toc367109572)

[Task 3: Customize Text Styles 22](#_Toc367109573)

[Task 4: Customize Data Templates 24](#_Toc367109574)

[Exercise 3: Search 26](#_Toc367109575)

[Task 1: Add Search Declaration 26](#_Toc367109576)

[Task 2: Add Search Results Page 27](#_Toc367109577)

[Task 3: Update Data Source to Return Search Results 29](#_Toc367109578)

[Task 4: Update Binding for Search Results Page 30](#_Toc367109579)

[Task 5: Update Search Results Page Styles 34](#_Toc367109580)

[Exercise 4: Share 37](#_Toc367109581)

[Task 1: Add Handler for Share Charm Invocation 37](#_Toc367109582)

[Task 2: Define the Content Structure to Share 40](#_Toc367109583)

[Exercise 5: App Bar 42](#_Toc367109584)

[Task 1: Add Bottom App Bar 42](#_Toc367109585)

[Task 2: Add Command to App Bar 42](#_Toc367109586)

[Exercise 6: Live Tiles 44](#_Toc367109587)

[Task 1: Choose the Type of Live Tile 44](#_Toc367109588)

[Task 2: Configure the Content for the Live Tile 45](#_Toc367109589)

[Task 3: Enable the Live Tile 45](#_Toc367109590)

[Optional Exercises 47](#_Toc367109591)

[Exercise 1: Semantic Zoom 48](#_Toc367109592)

[Task 1: Add Semantic Zoom control 48](#_Toc367109593)

[Task 2: Define the ZoomedOutView 49](#_Toc367109594)

[Exercise 2: Windowing Modes 51](#_Toc367109595)

[Task 1: Define the Minimum Width for the Application 51](#_Toc367109596)

[Task 2: Define the Narrower View for the Page 52](#_Toc367109597)

[Task 3: Set Up the Visual States 53](#_Toc367109598)

[Task 4: Handle the Window.SizeChanged Event 53](#_Toc367109599)

[Exercise 3: Secondary Tiles 56](#_Toc367109600)

[Task 1: Add the Pin To Start Button 56](#_Toc367109601)

[Task 2: Handle the Button Event 57](#_Toc367109602)

[Task 3: Enable Secondary Tile Pinning 57](#_Toc367109603)

[Task 4: Enable Secondary Tile Unpinning 60](#_Toc367109604)

[Task 5: Enable Entry Through the Secondary Tile 62](#_Toc367109605)

[Conclusion 63](#_Toc367109606)

[Lab Completion 64](#_Toc367109607)

[Achievement Summary 64](#_Toc367109608)

[Appendix 65](#_Toc367109609)

[Table of Figures 66](#_Toc367109610)

[Table of Code Samples 68](#_Toc367109611)

# Introduction

## **Overview**

### Welcome to the App Clinic Contoso Electronics Lab

One of three available for the **Windows 8.1 App Clinic[[1]](#footnote-2)** workshop, this lab provides an introduction to the main design and functional aspects of a Windows Store-type product catalog app. It shows you how to create a simple product hub using tiles, tied to details pages with associated product information.

You and your fellow attendees begin with a new project and go about the task of building the various pieces of the application. The solution uses code snippets that will help you rapidly develop the complete application within the scope of a 5-6 hour, instructor-led session.

#### About the labs

The App Clinic labs have a unique structure, in that participants are invited to choose one of three tracks for their development activities. The other two lab tracks, while providing a similar exploration of common Windows Store-style app features, are devoted to field service management and investment banking scenarios. The intent is to give all participants—who, after all, hail from diverse industries and have different business needs—a measure of choice in terms of the kind of app they will each spend the day learning how to build.

One or the other lab track may be more interesting or useful to you because its scenario may be directly or indirectly relevant to the kind of business you are in or the specific business problem you are trying to solve. Your business may need to manage and represent a catalog of goods or services to customers, and so the Contoso Electronics lab track is potentially a good choice for you. Or, perhaps your business is looking for ways to sharpen its field service operations, so the Field Engineer lab track may be the most sensible choice. Even if none of these scenarios exactly fits your circumstances, any one of the labs will give you a solid understanding of several common features of the modern Windows app style and fundamental practices you can take home and immediately turn to the purpose of developing well-designed, innovative apps that help drive business success.

### Objectives

The key objectives for the lab are three:

* Learn the fundamentals of developing a Windows 8.1, Windows Store-style app based on one of three models.
* Using your data and branding, build a working prototype app for your business.
* Gain a clear understanding of next steps and resources for further exploration.

### Lab Structure

After an introduction and overview of Windows 8.1 and demos of the full versions of the apps featured in the App Clinic, the lab will dive into the lab sessions, which will be a combination of hands-on exercises and presentations, as described in Table 1 below.

| Lab Topic | Duration | Type |
| --- | --- | --- |
| Data binding | 60 mins | HOL |
| Custom branding, start screen & logo | 60 mins | HOL |
| Search & search suggestions | 30 mins | HOL |
| Share | 30 mins | HOL |
| App bar | 30 mins | HOL |
| Semantic zoom | 15 mins | PPT |
| Snapped view | 15 mins | PPT |
| Live tiles | 30 mins | HOL |
| Secondary tiles & notifications | 30 mins | PPT |
| HOL = Hands-on-Lab | PPT = Presentation | | |

Table 1. Duration and type of topics treated in the lab

At the end of the day, once the exercises are complete, the remaining time—roughly an hour—will be spent on a discussion of best practices and a final wrap-up with next steps.

### Estimated Time for Completion

The full set of lab exercises require approximately 6 hours in an instructor-led session, with appropriate time built in for a mid-session break. On his/her own, an advanced student can conceivably complete the exercises in a much shorter period of time, depending on experience and/or aptitude.

# Getting Started

## System Requirements

This section provides an overview about the hardware and software requirements for this lab. This lab requires the **Windows 8.1 Preview** version, so please make sure that your PC is running this version of the Windows operating system before coming to this lab.

NOTE ▶ There are two ways to install Windows 8.1 Preview: either through the Windows Store or by using an ISO image file (see <http://windows.microsoft.com/en-us/windows-8/preview-download>).

* If you're running Windows 8 in one of the 13 languages for which the preview is available, we strongly recommend that you download the software from the Windows Store.
* If you're running Windows 8 and you want to install Windows 8.1 Preview using the ISO image file, it's important that you know the location of the Windows 8 product key. When you update to the final edition of Windows 8.1, you'll need your Windows 8 product key to activate.
* If you're running Windows XP, Windows Vista, or Windows 7, you can download the ISO image file and use it to create a bootable DVD or USB flash drive.

### Hardware

The following are the system requirements for Windows 8.1 Preview. Note that Windows 8.1 Preview works well on the same hardware that powers Windows 8.

* **Processor:** 1 gigahertz (GHz) or faster
* **RAM:** 1 gigabyte (GB) (32-bit) or 2 GB (64-bit)
* **Free hard disk space:** 16 GB (32-bit) or 20 GB (64-bit)
* **Graphics card:** Microsoft DirectX 9 graphics device with WDDM driver

NOTE ▶ In order to use Windows 8.1 Preview you must sign in to your PC with a Microsoft account. The option to create a local account will be made available with the final release of Windows 8.1.

### Software

To complete the lab exercises, at a minimum you need **Microsoft Visual Studio Express 2013 Preview for Windows** as your development tool. This tool requires Windows 8.1 Preview to be running on your PC; the system requirements are as stated above.

Visual Studio Express 2013 Preview for Windows provides the necessary tools for creating innovative and compelling Windows Store apps. Included are a full-featured code editor, a powerful debugger, a focused profiler, and rich language support that you can use to build apps that you've written in HTML5/JavaScript, C++, C#, or Visual Basic. It also includes a device simulator that you can use to test Windows Store apps on multiple kinds of devices.

NOTE ▶ You can download Microsoft Visual Studio Express 2013 Preview for Windows and find additional information about the software at <http://www.microsoft.com/en-us/download/details.aspx?id=39313>.

## Solution Overview

This section provides an overview of the solution that will be used throughout the lab exercises, covering project structure, application functionality, and navigation within the application. For this lab, you will be using for your template the Contoso Electronics application, a Windows Store-style business-to-consumer (B2C) app designed to help customers easily search, browse, discover, and purchase consumer electronics products online.

### Solution Structure

In this lab, you will be using the **AppClinic-ContosoElectronics** solution. You will find two sets of folders in the solution, representing the starting and completed states of the lab, respectively labeled “Start” and “Final,” as described below.

* **ContosoElectronics-Start:** Represents the initial state of the project to be used as the starting point for lab exercises.
* **ContosoElectronics-Final:** Represents the final state of the project to be used as the reference project.

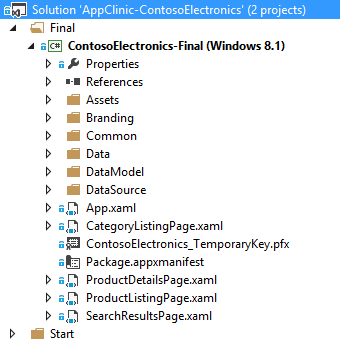


Figure 1. The structure of the Contoso Electronics solution

The key folders within the solution are described below in Table 2.

|  |  |
| --- | --- |
| Folder Name | Purpose/ContenTs |
| Assets | This folder is dedicated to keep all branding logos. Following items are placed inside this folder.   * 4 tile images (150 x 150, 310 x 150, 310 x 310, 70 x 70) * A store logo (50 x 50) * A splash-screen image (620 x 300) * A search logo (30 x 30) |
| Branding | For the custom theme images, custom styles, and custom template XAML files. See the “Branding Considerations” section below for more information. |
| Common | For the common code base, shared across the application. Converters, helpers, and custom types (not data models) are placed inside this folder. |
| Data | Dedicated to offline data elements (XML and artifacts). The following are placed inside this folder:   * Category.xml – List of product categories along with subcategories within each category * Product.xml – List of products * ProductImages folder – Images of all products * ProductCategoryImages folder – Images of all categories |
| DataModel | For all custom data model types. See the “Data Considerations” section below for more information. |
| DataSource | Contains a couple of sealed class definitions that help us to de-serialize offline data stored in the Data folder.  These class files act as data sources for the entire application. The static sync methods defined inside are accessed whenever needed. |
| *<Root folder>* | All screen XAMLs are placed in the root directory. You can also find the Package.appxmanifest and TemporaryKey files here. |
| *XAMLs* | * CategoryListingPage – First screen of the application, showing all the product categories and subcategories * ProductListingPage – Lists all the products within a subcategory * ProductDetailsPage – Provides all the details for a particular product |

Table 2. Contents of each of the solution folders

### Navigation Structure

The Contoso Electronics application uses a hierarchical system of navigation. This pattern, commonly used by Windows Store apps, provides a fast and fluid navigation. It is best for apps with large content collections or many distinct sections of content for a user to explore.

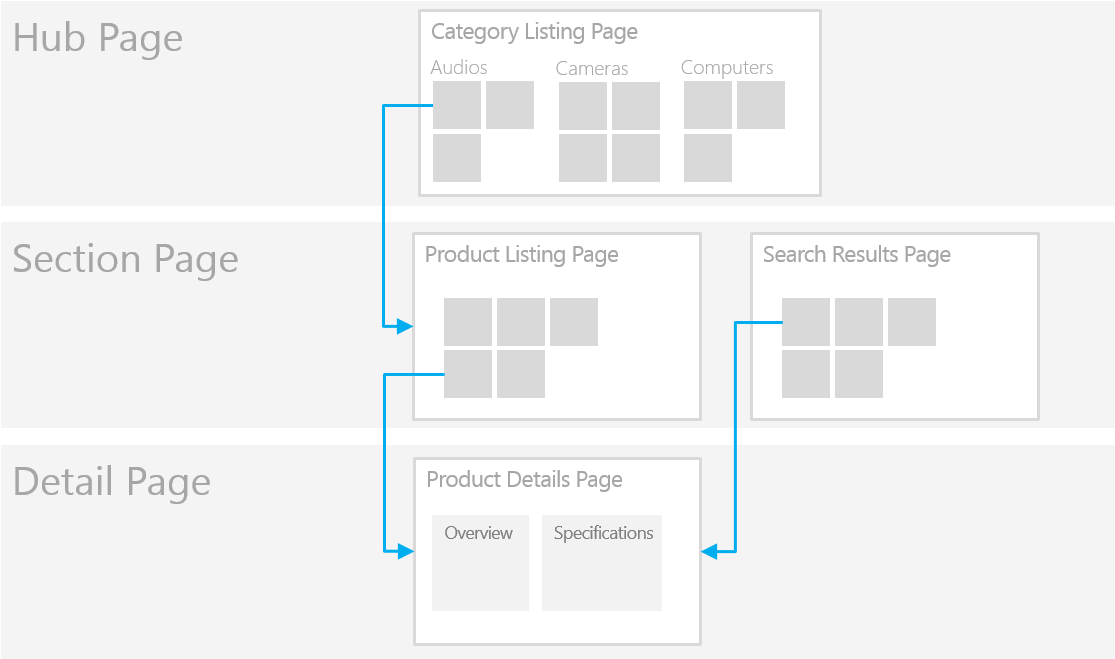


Figure 2. Simple wireframe showing how the hierarchical navigation system works

NOTE ▶ For further guidance regarding the various navigation patterns for Windows Store apps, see the [pertinent MSDN article](http://msdn.microsoft.com/en-us/library/windows/apps/hh761500.aspx#hierarchical_system).

### Getting to Know the Application

Before you start the lab exercise, take a moment to get familiar with the application. Just build and run the **ContosoElectronics-Final** project and closely inspect the application, including its Windows 8.1 platform features.

The figure below shows the welcome page, which lays out the various product categories and subcategories.

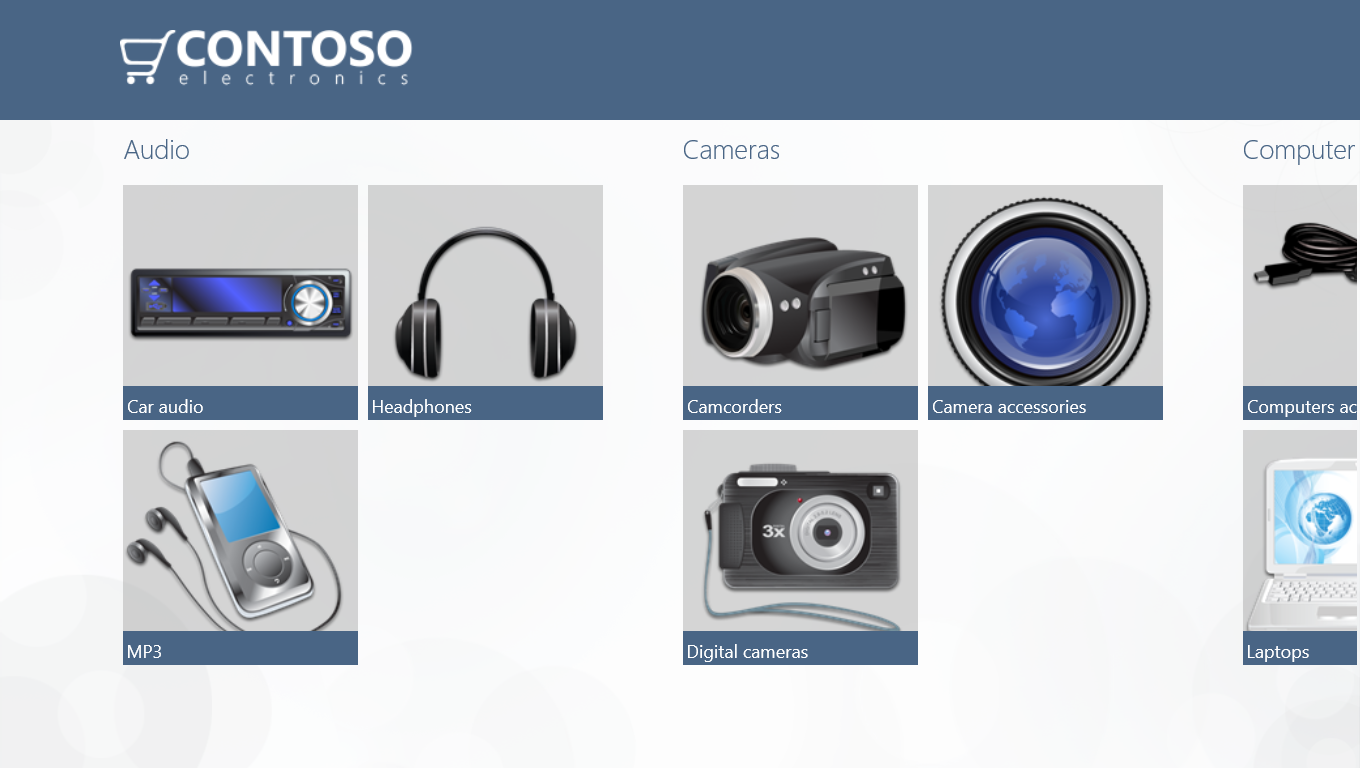


Figure 3. Welcome page for the Contoso Electronics app

#### Suggested activities

Take a moment to navigate throughout the application using the navigation structure described above. As you explore, perform the following tasks:

* Observe the live tile and the way it displays product information.
* Try out the various Start screen tile sizes.
* Try Semantic Zoom on the welcome page.
* Search for a product using the Search charm.
* Share a product details page using the Share charm.
* Pin a product on the Start screen from its details page.

## Branding Considerations

A brand is generally defined by the qualities for which a business wants to be known—its identity in the marketplace. When designing your Windows Store apps, you need to ensure that your apps incorporate the essence of your brand. This is both for the sake of overall consistency in the visual design of your organization’s assets—its “brand cohesion”—and the need to differentiate your apps from all the other apps available in the Windows Store, in particular those belonging to your competitors.

### Understanding Visual Elements

The expression of your brand is achieved through a set of visual elements—for example, a distinctive color palette, graphics, layout, and photography style. These elements work together to create a repeatable and recognizable visual system, consistently executed across a variety of media types and digital assets, such as broadcast, print, and Web marketing materials, Web sites, applications, signage, etc. Think of these visual elements as the knobs and dials you manipulate through code to give your Windows Store app a unique look-and-feel.

|  |  |
| --- | --- |
| VISUAL Element | Description |
| Colors | Color is a key attribute for expressing brand. Apply the primary color associated with your brand in ways that tell people that this app comes from your business. |
| Graphics | Use graphics to reinforce your brand by adding character to the presentation of content. Too many graphics, however, can interfere with the flow of your content and can become mere decoration or distraction. |
| Images | Illustrations and photography should also reflect your brand. Reuse the same imagery and style from your company's other media or websites. |
| Grid | The Windows Store app grid system helps unify the visual elements of your app's presentation. The grid aligns your branded app's UI to work with the rest of Windows. |
| Layout | The arrangement of visual elements for all pages needs to be relevant to your brand. You should also strive for consistency across page and content types. |
| Logo | Use your logo to help people quickly identify your app and recognize your brand. |
| Typography | Typefaces are a key part of Windows Store apps. The right choice can be as impactful to your brand as color, logo, or layout, so be thoughtful about the typography you use. |

Table 3. Visual elements described

### Image Assets/Logos

A Windows Store app includes a basic set of image assets—such as logos or representative icons—for which there are some basic requirements and considerations to bear in mind. The assets specifically required for this lab are described below.

| IMAGE ASSETS | SIZE | DESCRIPTION |
| --- | --- | --- |
| Logo – Small Tile | 70 x 70 | For the **small** square tile image of the app in the Start screen. |
| Logo – Medium Tile | 150 x 150 | For the **medium** square tile image of the app in the Start screen. |
| Logo – Large Tile | 310 x 310 | For the **large** square tile image of the app in the Start screen. |
| Logo – Wide Tile | 310 x 150 | For the **wide** tile image of the app in the Start screen. While not strictly required, without it, a user will be unable to resize the tile to a wide format, nor benefit from the additional detail available in wide-formatted notifications. |
| Small Logo | 30 x 30 | Displayed with your app's display name in search results returned on the Start screen; also used in the list of searchable apps and in the zoomed-out view of the Start screen. |
| Store Logo | 50 x 50 | For the Windows Store when it displays your app's listing in search results and together with the app's description in the listing page. |
| Badge Logo | 24 x 24 | Exclusively used for display next to the badge notification to identify the app on the lock screen; must be monochromatic. |
| Splash Screen Image | 620 x 300 | Shown on your app’s splash screen, which appears for a short interval when the user first opens the app. A mandatory element for which you can customize the background color. |

Table 4. Image assets required for the lab

NOTE ▶ Bear in mind the following considerations as you design and implement your image assets:

* All the above images assets/logos are defined within the Package.appxmanifest file of the project. The **Visual Assets** section of the **Application UI** tab provides a list of all of these image assets.
* Make sure that the various image assets match the exact size specifications defined in the table above.
* To provide a better experience on computers with different screen resolutions, you should consider including scaled versions of the images in your app. Note that the scaled images should be designed for the image size and not simply resized from the original image.
* When designing the images for the Start screen tiles, consider whether you want to include your app's name in the image or have Windows overlay the name on top of the image (see the [MSDN article](http://msdn.microsoft.com/en-us/library/windows/apps/hh694077.aspx) for further information).

### Understanding XAML Styling

It is important to understand how styling works for XAML applications and what you need to do to customize the styles to incorporate your own branding. This is where the **ResourceDictionary** class and the **ThemeDictionaries** property of that class come into play.

#### The ResourceDictionary class

The **ResourceDictionary** contains resources used by components of the app. The purpose of a **ResourceDictionary** is to enable you to define resources in XAML, and then retrieve them through XAML references made with either the [StaticResource](http://msdn.microsoft.com/en-us/library/windows/apps/hh758287.aspx) markup extension or the [ThemeResource](http://msdn.microsoft.com/en-us/library/windows/apps/dn263118.aspx) markup extension. In this way, you can:

* Reuse resources you've already defined once in XAML.
* Control the complexity of your XAML by following best practices for how to factor XAML-defined elements and any shared value
* Ensure that certain values such as brush colors or pixel measurements are used consistently.

#### The ThemeDictionaries property

**ThemeDictionaries** define a collection of merged resource dictionaries that are specifically keyed and composed to address theme scenarios. Each **resource dictionary** within **ThemeDictionaries** contain resources for a specific theme and must have an **x:Key** attribute representing the name of that theme.

The example below represents three resource dictionaries with three different themes. Note that each resource dictionary has a unique **x:Key** attribute, while the **SolidColorBrush** resource has the same **x:Key** attribute across the three resource dictionaries, but with a different color value specific to each theme.

|  |
| --- |
| XML |
| * + 1. <ResourceDictionary     2. xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"     3. xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">     4. <!—Theme Dictionaries contains resources that vary across themes -->     5. <ResourceDictionary.ThemeDictionaries>     7. <!-- Contains Default theme resources -->     8. <ResourceDictionary x:Key="Default">     9. <SolidColorBrush x:Key="ApplicationForegroundThemeBrush" Color="#DEFFFFFF" />     10. ...     11. </ResourceDictionary>     12. <!-- Contains High Contrast theme resources -->     13. <ResourceDictionary x:Key="HighContrast">     14. <SolidColorBrush x:Key="ApplicationForegroundThemeBrush" Color="#FF000000" />     15. ...     16. </ResourceDictionary>     17. <!-- Contains Light theme resources -->     18. <ResourceDictionary x:Key="Light">     19. <SolidColorBrush x:Key="ApplicationForegroundThemeBrush" Color="#FFCCCCCC" />     20. ...     21. </ResourceDictionary>     22. </ResourceDictionary.ThemeDictionaries>     23. ...     24. </ResourceDictionary> |

Code Sample 1. Resource dictionaries and theme dictionaries

NOTE ▶ In Windows 8.1, all the standard XAML resources have been moved out of the project template and are now maintained as part of the platform. All resources including brush colors, pixel measurements, control styles, etc. have been moved to a file called Generic.xaml stored in C:\Program Files (x86)\Windows Kits\8.1\Include\Winrt\Xaml\Design.

## Data Considerations

Without the requisite data, a well-designed app is just an empty—albeit pretty—shell. Indeed, as a practical matter, an app can only be as good as the data it’s intended to showcase. Accordingly, success with this lab depends on understanding the offline data and data model classes for the application and how they are organized.

### Data Model

The data for the app finds its structure in the DataModel folder. Below is the list of available classes.

|  |  |  |
| --- | --- | --- |
| Class Name | Location | Purpose/Notes |
| ProductCategory | /DataModel/ProductCategory.cs | This class represents a product category and its properties. It contains a list of subcategories. |
| ProductSubCategory | /DataModel/ProductSubCategory.cs | Represents a subcategory within a category. |
| Product | /DataModel/Product.cs | Represents a single product. Category IDs and subcategory IDs denote the categories and subcategories to which it belongs. |
| ProductSpecification | /DataModel/ProductSpecification.cs | Contains a name-value pair of a single product specification. Each product possesses a list of such specifications. |

Table 5. Data classes in the Contoso Electronics data model

To help you visualize the relationship of the different classes in the model, a class diagram is provided below.

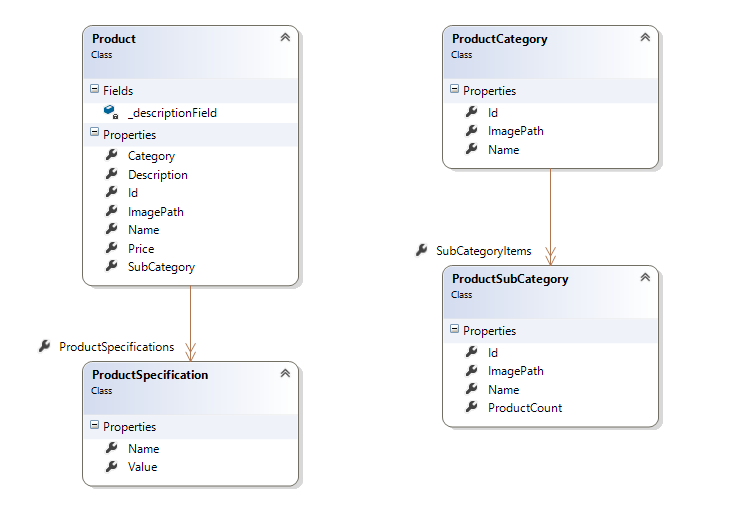


Figure 4. Class diagram of the Contoso Electronics data model

### Data Schema

The Contoso Electronics application uses XML files as the containers for offline data, and can be found in the Data folder. There are two XML files used in this lab, Category.xml and Product.xml, as described in the table below.

|  |  |
| --- | --- |
| File name | Purpose |
| Category.xml | Keeps the list of product categories. Each category contains a list of subcategories. |
| Product.xml | Keeps the list of all products with all the details for each product. |

Table 6. Schema descriptions

The XML schema for each is found below.

#### Category.xml schema

|  |
| --- |
| XML |
| * + 1. <ArrayOfProductCategory xmlns="http://schemas.datacontract.org/2004/07/ContosoElectronics.DataModel" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">     2. <ProductCategory>     3. <Id>1</Id>     4. <ImagePath/>     5. <Name>Audio</Name>     6. <SubCategoryItems>     7. <ProductSubCategory>     8. <Id>11</Id>     9. <ImagePath>/Data/ProductCategoryImages/4040101.png</ImagePath>     10. <Name>Car audio</Name>     11. <ProductCount>13</ProductCount>     12. </ProductSubCategory>     13. <ProductSubCategory>     14. <Id>12</Id>     15. <ImagePath>/Data/ProductCategoryImages/4010101.png</ImagePath>     16. <Name>MP3</Name>     17. <ProductCount>34</ProductCount>     18. </ProductSubCategory>     19. <ProductSubCategory>     20. <Id>13</Id>     21. <ImagePath>/Data/ProductCategoryImages/4030101.png</ImagePath>     22. <Name>Headphones</Name>     23. <ProductCount>31</ProductCount>     24. </ProductSubCategory>     25. </SubCategoryItems>     26. </ProductCategory>     27. ...     28. </ArrayOfProductCategory> |

Code Sample 2. The Category.xml schema

#### Product.xml schema

|  |
| --- |
| XML |
| * + 1. <ArrayOfProduct xmlns="http://schemas.datacontract.org/2004/07/ContosoElectronics.DataModel" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">     2. <Product>     3. <Category>1</Category>     4. <Description>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat</Description>     5. <Id>5637146791</Id>     6. <ImagePath>/Data/ProductImages/4040102.png</ImagePath>     7. <Name>Proseware 50W Car Radio</Name>     8. <Price>89.99</Price>     9. <ProductSpecifications>     10. <ProductSpecification>     11. <Name>HardDrive(GB)</Name>     12. <Value>2</Value>     13. </ProductSpecification>     14. <ProductSpecification>     15. <Name>Connector</Name>     16. <Value>3.5mm TRS</Value>     17. </ProductSpecification>     18. <ProductSpecification>     19. <Name>Noise Cancelling</Name>     20. <Value>Yes</Value>     21. </ProductSpecification>     22. <ProductSpecification>     23. <Name>Music Controls</Name>     24. <Value>Yes</Value>     25. </ProductSpecification>     26. <ProductSpecification>     27. <Name>USB Support</Name>     28. <Value>Yes</Value>     29. </ProductSpecification>     30. <ProductSpecification>     31. <Name>Folding</Name>     32. <Value>Yes</Value>     33. </ProductSpecification>     34. <ProductSpecification>     35. <Name>Cord Length</Name>     36. <Value>2.4m</Value>     37. </ProductSpecification>     38. </ProductSpecifications>     39. <SubCategory>11</SubCategory>     40. </Product>     41. ...     42. </ArrayOfProduct> |

Code Sample 3. The Product.xml schema

# Lab Exercises

## Exercise 1: Local Data Binding

In this exercise, you will setup your custom local data for the Windows Store application. This includes copying your XML & image files representing the custom data, setting up the data source, and finaly binding the data to the pages. (Refer to the ***Data Considerations***section above before getting started with this exercise.)

This exercise primarily focuses on the product categories schema to illustrate the data binding within the application. There are three tasks in the exercise:

1. Set up the data XML files.
2. Define the data source.
3. Fetch and Bind the data.

Just follow the steps for each task and use the code samples provided, wherever applicable.

### Task 1: Set Up the Data XML files

1. As a first step, review the existing folders & files associated with the data. Refer to the figure below that illustrates the folder structure with all the files related to data binding.

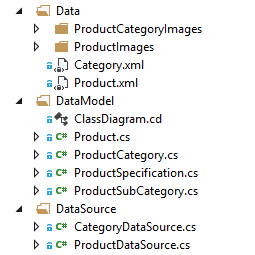


Figure 5. Folders and files related to data binding

1. Review your custom data XML files as well images. Make sure to update the data model classes within the DataModel folder if there are any changes to the XML schema.
2. Replace the Category.xml file within the Data folder with your updated XML file. In case you don’t have any updated file, then use the sample Category.xml file provided with the project.

### Task 2: Define the Data Source

1. Open the class file CategoryDataSource.cs within the DataSource folder and review the existing code.
2. Add a private method **ReadXmlDataFromLocalStorageAsync** within this class to fetch the XML data from the package. Refer to the code sample below.

|  |
| --- |
| C# |
| * + 1. private async Task ReadXmlDataFromLocalStorageAsync()     2. {     3. // Return if data is already loaded     4. if (\_categoryDataSource.AllCategories != null)     5. return;     6. try     7. {     8. var dataFolder = await Package.Current.InstalledLocation.GetFolderAsync("Data");     9. StorageFile sessionFile = await dataFolder.GetFileAsync("Category.xml");     10. using (IRandomAccessStreamWithContentType sessionInputStream = await sessionFile.OpenReadAsync())     11. {     12. var sessionSerializer = new DataContractSerializer(typeof(ProductCategory[]));     13. var restoredData = (ProductCategory[])sessionSerializer.ReadObject(sessionInputStream.AsStreamForRead());     14. \_allCategories = (List<ProductCategory>)restoredData.ToList();     15. }     16. }     17. catch (Exception ex)     18. {     19. \_allCategories = null;     20. }     21. } |

Code Sample 4. Add private method to fetch XML data

1. Add a public method **GetListAsync** within the same class to fetch the list of product categories to be displayed on the first screen of the application.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// Gets the category list.     3. /// </summary>     4. public static async Task<List<ProductCategory>> GetListAsync()     5. {     6. await \_categoryDataSource.ReadXmlDataFromLocalStorageAsync();     7. return \_categoryDataSource.AllCategories;     8. } |

Code Sample 5. Add public method to fetch list of categories

1. Add another public method **GetCategoryDetailsAsync** to fetch the category details for a given category name. Refer to the code sample below.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// Gets the category details.     3. /// </summary>     4. public static async Task<ProductCategory> GetCategoryDetailsAsync(string categoryName)     5. {     6. await \_categoryDataSource.ReadXmlDataFromLocalStorageAsync();     7. return \_categoryDataSource.AllCategories.Where(x => x.Name.ToUpper().StartsWith(categoryName.ToUpper())).FirstOrDefault();     8. } |

Code Sample 6. Add public method to fetch category details

1. Add one more public method **GetSubCategoryNameAsync** to fetch the subcategory name based on its ID.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// Gets the Sub category Name.     3. /// </summary>     4. public static async Task<String> GetSubCategoryNameAsync(string subCategoryId)     5. {     6. await \_categoryDataSource.ReadXmlDataFromLocalStorageAsync();     7. return \_categoryDataSource.AllCategories.SelectMany(x => x.SubCategoryItems).FirstOrDefault(x => x.Id == subCategoryId).Name;     8. } |

Code Sample 7. Add public method to fetch subcategory name

### Task 3: Fetch and Bind the Data

The next task is to fetch the product category data from the category datasource using the methods we added within the previous task. After fetching the data, we will bind that that data to the various pages within the application.

1. Open the CategoryListingPage.xaml.cs class file and add the following lines of code within the **LoadState** method. Run the application after making this change to verify that the category data from your updated XML file is being reflected within the application.

|  |
| --- |
| C# |
| * + 1. // Fetch the Product Categories and assign them as the items source for the gridview displaying all Categories     2. ObservableCollection<ProductCategory> allGroups = new ObservableCollection<ProductCategory>(await CategoryDataSource.GetListAsync());     3. groupedItemsViewSource.Source = allGroups; |

Code Sample 8. Bind the product categories

1. Open the ProductListingPage.xaml.cs class file and add the following lines of code within the **LoadState** method. Run the application after making this change to verify that the category name is shown as the page header for the product listing page.

|  |
| --- |
| C# |
| * + 1. this.defaultViewModel["PageName"] = await CategoryDataSource.GetSubCategoryNameAsync(e.NavigationParameter.ToString()); |

Code Sample 9. Bind subcategory names

## Exercise 2: Branding

In this exercise, you will customize the branding for the Windows Store application. Branding includes configuring the logos for the various tile sizes, the search and store logos, the background and foreground colors for the application, customizing the text styles and the data templates, etc. (Refer to the ***Error! Reference source not found.*** section above before getting started with this exercise.)

There are four tasks in this exercise:

1. Configure visual assets (tiles & logos).
2. Customize background and foreground colors.
3. Customize text styles.
4. Customize data templates.

### Task 1: Configure Visual Assets

First, we’ll configure the visual assets for the application, including the tile images and logos, badge logo, splash screen, etc.

1. Open the Assets folder and review the various images/logos within that folder. Replace any/all of these files with your custom images/logos.

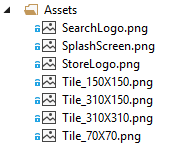


Figure 6. Contents of the Assets folder

* Make sure that your custom images/logos are as per the size specifications of the visual assets. Refer to Table 4 above for more information.

NOTE ▶ For additional guidance around size specifications, refer to the link below. <http://msdn.microsoft.com/en-us/library/windows/apps/hh846296.aspx#storelogo>.

* Make sure you keep the file names the same when your override the files with your custom images/logos. Should the file names be different, you will need to upate the file names within the application manifest file, as explained in the next step.

1. Open the Package.appxmanifest file in the folder root and refer to the **Application UI** tab, as shown below.

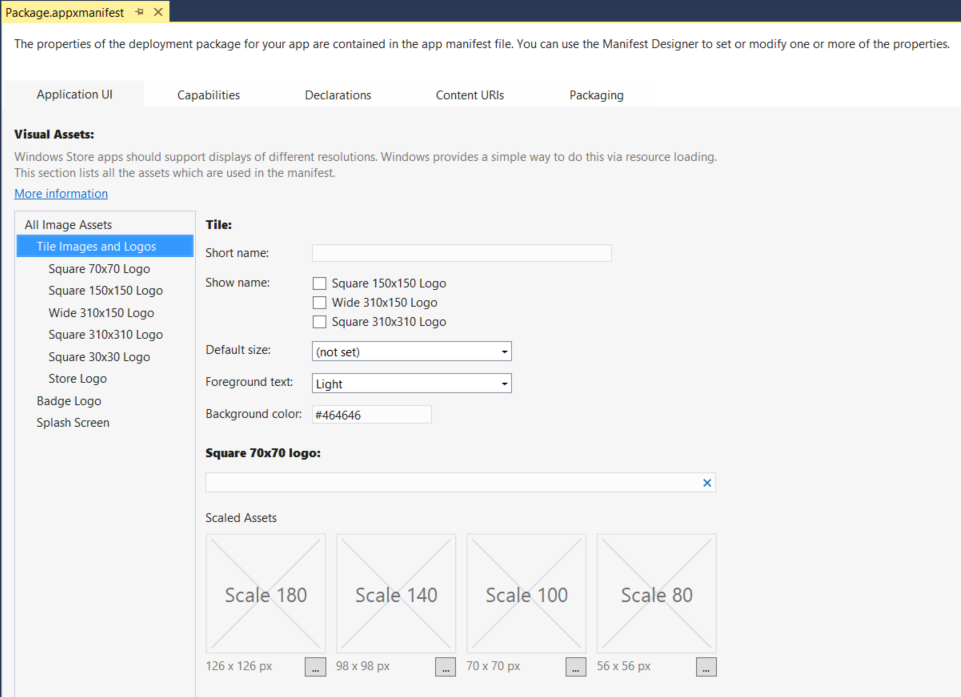


Figure 7. Configure image asset sizes

### Task 2: Customize Background & Foreground Colors

The next thing to do is to customize the primary and secondary colors for the entire application. This includes setting the background and foreground colors for various UI elements. In this task, we will override some of the key brush colors to reflect your custom branding.

Windows 8.1 provides standard brush colors as XAML resources that defines the primary and secondary colors for an application. These XAML resources are maintained as part of the platform and are stored in a file called **Generic.xaml** located in C:\Program Files (x86)\Windows Kits\8.1\Include\Winrt\Xaml\Design.

1. Open Generic.xaml and take a quick look at some of the standard brush colors defined within the theme dictionaries. Note that there are more than 240 brush colors that are defined for each theme.
2. Open the CustomStyles.xaml file within the Branding folder and locate the section shown below. This section represents a brief description about each color brush. Replace these colors with those that match your branding.

|  |
| --- |
| XAML |
| * + 1. <!--     2. ========================================================================     3. OVERRIDE STANDARD SYSTEM COLOR BRUSH     4. The below are some of the key system defined color brush resources that     5. can be overriden to reflect your branding colors.     7. > AppBarBackgroundThemeBrush: Background Color for the App Bar     8. > AppBarBorderThemeBrush: Border Color for the App Bar     9. > AppBarItemForegroundThemeBrush: Foreground Color for App Bar Items     10. > ApplicationForegroundThemeBrush: Primary Foreground Color     11. > ApplicationHeaderForegroundThemeBrush: Foreground Color for Header Text     12. > ApplicationSecondaryForegroundThemeBrush: Secondary Foreground Color     13. > ApplicationPageBackgroundThemeBrush: Page Background Color     14. > BackButtonBackgroundThemeBrush: Background Color for Back Button     15. > BackButtonForegroundThemeBrush: Foreground Color for Back Button     16. ========================================================================     17. -->     18. <SolidColorBrush x:Key="AppBarBackgroundThemeBrush" Color="#99496585" />     19. <SolidColorBrush x:Key="AppBarBorderThemeBrush" Color="#FF000000" />     20. <SolidColorBrush x:Key="AppBarItemForegroundThemeBrush" Color="#FFFFFFFF" />     21. <SolidColorBrush x:Key="ApplicationForegroundThemeBrush" Color="#496585" />     22. <SolidColorBrush x:Key="ApplicationHeaderForegroundThemeBrush" Color="#FFFFFFFF" />     23. <SolidColorBrush x:Key="ApplicationSecondaryForegroundThemeBrush" Color="#99000000" />     24. <SolidColorBrush x:Key="BackButtonBackgroundThemeBrush" Color="Transparent" />     25. <SolidColorBrush x:Key="BackButtonForegroundThemeBrush" Color="#FFFFFFFF" /> |

Code Sample 10. Overriding default SolidColorBrush resources

1. Open the CustomStyles.xaml file within the Branding folder and locate the section shown below. This section represents a brief description about each custom color and image brush. These color brushes are not system colors but are added for some specific purpose like the background of tiles, page background image, etc. You can replace these colors with those that match your branding.

|  |
| --- |
| XAML |
| * + 1. <!--     2. ========================================================================     3. DEFINE NEW CUSTOM STYLES:     5. PageBackgroundImage: Background Image for all the pages     6. PageHeaderBackgroundBrush: Background Color for the Page Title Header     7. TileBackgroundBrush: Background Color for active Tiles (clickable)     8. InactiveTileBackgroundBrush: Background Color for in-active Tiles     9. ========================================================================     10. -->     11. <ImageBrush x:Key="PageBackgroundImageBrush" ImageSource="/Branding/Background.png" />     12. <SolidColorBrush x:Key="PageHeaderBackgroundBrush" Color="#496585" />     13. <SolidColorBrush x:Key="TileBackgroundBrush" Color="#50496585" />     14. <SolidColorBrush x:Key="InactiveTileBackgroundBrush" Color="#10496585" /> |

Code Sample 11. Adding new SolidColorBrush and ImageBrush resources

### Task 3: Customize Text Styles

We’ll now customize the text styles for the entire application, including the font size, font weight, and the foreground colors for the various types of text that appear in the app.

Windows 8.1 provides standard text block styles as XAML resources that are maintained as part of the platform in a file called **Generic.xaml** located in C:\Program Files (x86)\Windows Kits\8.1\Include\Winrt\Xaml\Design.

1. Open **Generic.xaml** and take a quick look at the various text block styles found near the end of this file. Note that there are six standard text block styles shown below that are defined as part of the platform.

|  |
| --- |
| XAML |
| * + 1. <Style x:Key="HeaderTextBlockStyle" TargetType="TextBlock" BasedOn="{StaticResource BaseTextBlockStyle}">     2. <Setter Property="FontSize" Value="56"/>     3. <Setter Property="FontWeight" Value="Light"/>     4. <Setter Property="LineHeight" Value="40"/>     5. </Style>     6. <Style x:Key="SubheaderTextBlockStyle" TargetType="TextBlock" BasedOn="{StaticResource BaseTextBlockStyle}">     7. <Setter Property="FontSize" Value="26.667"/>     8. <Setter Property="FontWeight" Value="Light"/>     9. <Setter Property="LineHeight" Value="30"/>     10. </Style>     11. <Style x:Key="TitleTextBlockStyle" TargetType="TextBlock" BasedOn="{StaticResource BaseTextBlockStyle}">     12. <Setter Property="FontWeight" Value="SemiBold"/>     13. </Style>     14. <Style x:Key="SubtitleTextBlockStyle" TargetType="TextBlock" BasedOn="{StaticResource BaseTextBlockStyle}">     15. <Setter Property="FontWeight" Value="Normal"/>     16. </Style>     17. <Style x:Key="BodyTextBlockStyle" TargetType="TextBlock" BasedOn="{StaticResource BaseTextBlockStyle}">     18. <Setter Property="FontWeight" Value="SemiLight"/>     19. </Style>     20. <Style x:Key="CaptionTextBlockStyle" TargetType="TextBlock" BasedOn="{StaticResource BaseTextBlockStyle}">     21. <Setter Property="FontSize" Value="12"/>     22. <Setter Property="FontWeight" Value="Normal"/>     23. </Style> |

Code Sample 12. Standard framework TextBlock styles

1. Next, open CustomStyles.xaml in the Branding folder and locate the section shown below. This section presents a brief description of each text block style. Customize these styles to match your branding.

|  |
| --- |
| XAML |
| * + 1. <!--     2. ========================================================================     3. OVERRIDE STANDARD SYSTEM TEXT BLOCK STYLES     4. The below are the key system defined text bloxk styles that can be     5. overriden to reflect your branding colors.     7. > HeaderTextBlockStyle: Style for the Page Header Text     8. > SubheaderTextBlockStyle: Style for the Group/Section Header Text     9. > TitleTextBlockStyle: Style for the Title text within a section     10. > SubtitleTextBlockStyle: Style for the sub title text within a section     11. > BodyTextBlockStyle: Style for the body text     12. > CaptionTextBlockStyle: Style for the caption text     13. ========================================================================     14. -->     15. <Style x:Key="HeaderTextBlockStyle" TargetType="TextBlock" BasedOn="{StaticResource BaseTextBlockStyle}">     16. <Setter Property="FontSize" Value="56"/>     17. <Setter Property="FontWeight" Value="Light"/>     18. <Setter Property="LineHeight" Value="40"/>     19. <Setter Property="Foreground" Value="{StaticResource ApplicationHeaderForegroundThemeBrush}"/>     20. </Style>     21. ... |

Code Sample 13. Overriding framework TextBlock styles

### Task 4: Customize Data Templates

Lastly for this exercise, we need to customize the structure, layout, and overall design of the data templates, of which there are two for this lab:

* Subcategory item template
* Product item template

Let’s take these in order.

1. **The subcategory item template.** This data template provides for the structure and layout for subcategory items, including the data fields shown in the figure below. This data template is used to show subcategories witihin the grouped categories in the product catalog hub.



Figure 8. Data template for subcategory items

Customize the style elements for this template—alignments, font sizes, etc.—in accordance with your brand guidelines and/or the overall design you wish your app to have.

|  |
| --- |
| XAML |
| * + 1. <DataTemplate x:Key="ProductSubCategoryTemplate">     2. <Grid HorizontalAlignment="Left" Width="235" Height="235" Margin="0" Background="#66999999">     3. <Image Source="{Binding ImagePath}" Width="235" Height="235" Stretch="Uniform" VerticalAlignment="Top"/>     4. <StackPanel VerticalAlignment="Bottom" Background="#496585" Orientation="Horizontal">     5. <TextBlock Text="{Binding Name}" Style="{StaticResource TitleTextBlockStyle}" Foreground="White" TextTrimming="WordEllipsis" TextWrapping="NoWrap" Margin="5,7,5,7" VerticalAlignment="Center" HorizontalAlignment="Center"/>     6. </StackPanel>     7. </Grid>     8. </DataTemplate> |

Code Sample 14. Customize styles in the subcategory item data template

1. **The product item template.** This data template establishes the structure and layout for the product items, including the data fields as shown in the figure immediately below. This data template is used to show the product items on the product listing and search results pages.

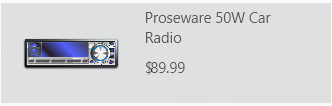
****

Figure 9. Data template for product items

Just as above, customize the style elements for this template—alignments, font sizes, etc.—in accordance with your brand guidelines and/or the overall design you wish your app to have.

|  |
| --- |
| XAML |
| * + 1. <DataTemplate x:Key="ProductItemTemplate">     2. <Grid Height="100" Width="330" Margin="0,0,0,0" Background="#44999999" HorizontalAlignment="Left" VerticalAlignment="Top">     3. <Grid.ColumnDefinitions>     4. <ColumnDefinition Width="Auto"/>     5. <ColumnDefinition Width="\*"/>     6. </Grid.ColumnDefinitions>     7. <Grid.RowDefinitions>     8. <RowDefinition Height="\*"/>     9. <RowDefinition Height="40"/>     10. </Grid.RowDefinitions>     11. <Border Grid.RowSpan="2" Width="140" Height="100">     12. <Image Source="{Binding ImagePath}" Stretch="Uniform"/>     13. </Border>     14. <StackPanel Grid.Column="1" Grid.RowSpan="2" HorizontalAlignment="Left" Orientation="Vertical" VerticalAlignment="Top" Margin="5,0,0,0">     15. <TextBlock Text="{Binding Name}" Style="{StaticResource SubtitleTextBlockStyle}" TextWrapping="Wrap" MaxWidth="150"/>     16. <StackPanel Orientation="Horizontal" Margin="0,10,0,0">     17. <TextBlock Text="$" Style="{StaticResource SubtitleTextBlockStyle}" />     18. <TextBlock Text="{Binding Price, Converter={StaticResource PriceConverter}, ConverterParameter={Binding Price} , ConverterLanguage=English}" Style="{StaticResource SubtitleTextBlockStyle}" MaxHeight="80" TextTrimming="WordEllipsis" TextWrapping="Wrap"/>     19. </StackPanel>     20. </StackPanel>     21. </Grid>     22. </DataTemplate> |

Code Sample 15. Customize styles in the product item data template

## Exercise 3: Search

Windows Store apps expose search consistently. Swiping from the right edge of the screen or pressing the Windows+C key combination invokes the charm bar. The Search charm can be either context-sensitive or global. When you invoke it from the desktop or the Start screen, you’ll access the system-wide search functionality. Note that the UI and actions required to invoke search are the same for all the apps.

NOTE ▶ This lab uses the Windows 8-style search, as described in this exercise. Windows 8.1 exposes search differently, privileging the implementation of an in-app search box rather than an invocation of search through the charm bar. Regardless, the code for implementing either is simple and consistent, and you should have little difficulty in transitioning from one style to the other.

In this exercise, you will learn how to implement the Search contract for the Contoso Financial Services application. You will configure the application to allow its users to perform a free-text search for products based on product ID, product name, etc.

There are five tasks in this exercise:

1. Add search declaration.
2. Add search results page.
3. Update data source to return search results.
4. Update binding for search results page.
5. Update styling for search results page.

### Task 1: Add Search Declaration

Before we start this task, just build and run the application and then bring up the Search charm. You will notice that the scope selector within the Search charm does not show the Contoso Electronics application yet, as shown in the figure below.

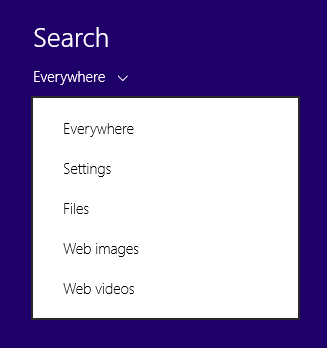


Figure 10. Search charm scope selector

The first task, then, is to add the search declaration so that the Contoso Electronics app starts showing up within the scope selector of the Search charm. Follow the steps below to add the search declaration.

1. Open the Package.appxmanifest file from the Solution Explorer and then select the **Declarations** tab.
2. Expand the **Available Declarations** drop-down list box and click **Search**.
3. Click **Add** and then save and close the file. Refer to the figure below, showing the search declaration added to the supported declarations for the app.

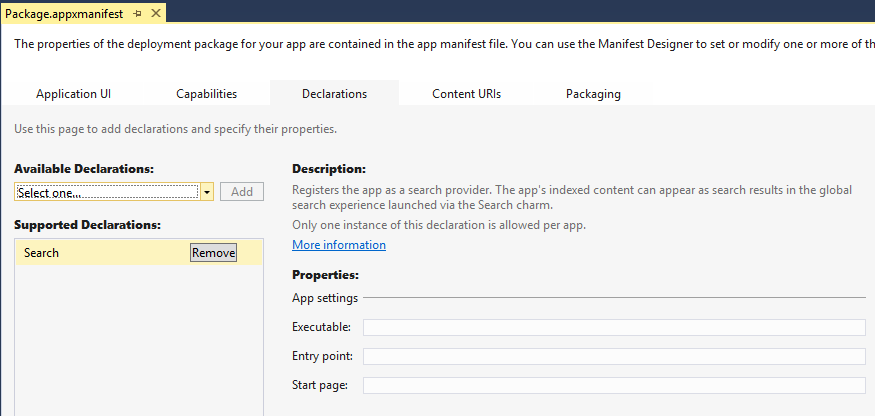


Figure 11. Search declaration

1. At this point, build and run the application again and then evoke the Search charm. You will notice that the scope selector of the Search charm now shows the Contoso Electronics app.

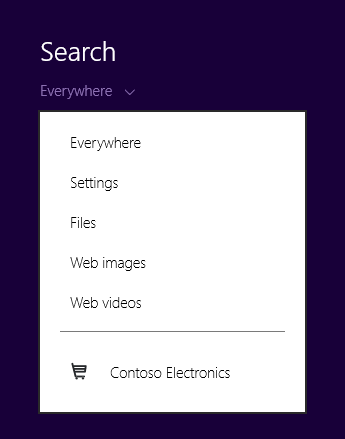


Figure 12. Search charm scope selector now displaying the app

1. Select **Contoso Electronics** in the scope selector of the Search charm and try to perform a search using any keyword. You will notice that nothing happens. This is because we have yet to create a search results page.

### Task 2: Add Search Results Page

The next task is to add a search results page that will show the search results for a given search query. The search results will comprise a list of products matching the search criteria. Note that Visual Studio 2013 provides a default template for the search results page. Follow the steps below to add the search results page.

1. In the Solution Explorer, right-click the project **ContosoFinancialServices-Start** and then select **Add New Item**. You should now see the Add New Item dialog box, as shown below.
2. In the left pane, select **Windows Store**, which appears in the **Visual C#** list. This will show only the Windows Store-specific templates.
3. Scroll down the list of templates and select **Search Results Page**. Change the name of the file to “SearchResultsPage.xaml” in the **Name** box and click **Add**.

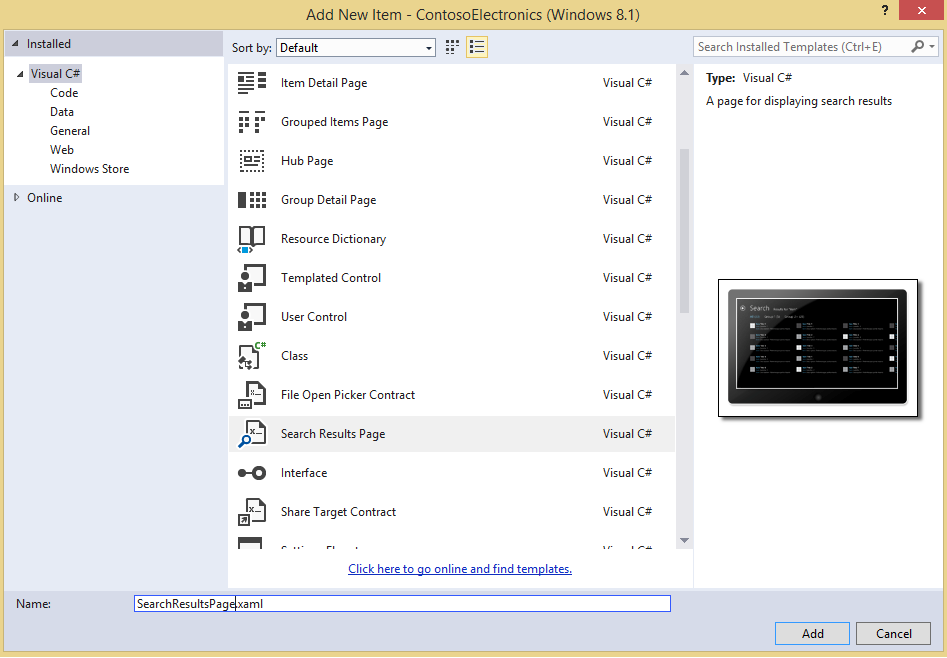


Figure 13. Add search results page

1. The next step to integrate the Search Results page within the application so that whenever the search is activated, the application redirects to this page. Open the application code-behind file App.xaml.cs and override the **OnSearchActivated** method. This method is invoked whenever the application is activated to perform a search.
2. Navigate to **Search Results Page** within this method. Refer to the sample code below. Just copy-and-paste the code below within the App.xaml.cs file.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// Invoked when the application is activated to display search results.     3. /// </summary>     4. /// <param name="args">Details about the activation request.</param>     5. protected override void OnSearchActivated(SearchActivatedEventArgs args)     6. {     7. //Get the frame i.e. the content of the current window     8. Frame frame = Window.Current.Content as Frame;     9. //Navigate to the Search Results Page     10. frame.Navigate(typeof(SearchResultsPage), args.QueryText);     11. } |

Code Sample 16. Handler method activated on search

1. Now, build and run the application again and evoke the Search charm. Select the app **Contoso Financial Services** using the scope selector of the Search charm and try to perform a search using any keyword. You will notice that you are being redirected to the search results page, as shown below.



Figure 14. Search results page (default view)

### Task 3: Update Data Source to Return Search Results

This task involves updating the existing data source to fetch the search results based upon a search keyword. Here, we will implement the search functionality only for products, so we will need to update the existing data source for products.

1. Open the ProductDataSource.cs class file within the DataSource folder.
2. Add a new method **SearchProductsAsync** within this class. This method will accept the search keyword string as the input parameter and will perform the search within the list of jobs based on product name or product description. Finally, this method will return a filtered list of products matching the search criteria.
3. Refer to the sample code below, and copy-and-paste it into the ProductDataSource.cs file.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method searches the products by search text. The search text can be a part of the     3. /// Product name or Product description     4. /// </summary>     5. /// <param name="searchText">The search text.</param>     6. /// <returns>Search results</returns>     7. public static async Task<List<Product>> SearchProductsAsync(string searchText)     8. {     9. await \_productDataSource.ReadXmlDataFromLocalStorageAsync();     10. return     11. \_productDataSource.AllProducts.Where(     12. item =>     13. item.Name.ToUpper().Contains(searchText.ToUpper()) ||     14. item.Description.ToUpper().Contains(searchText.ToUpper())).ToList();     15. } |

Code Sample 17. Search product data source method

### Task 4: Update Binding for Search Results Page

The next task is to bind the search results page with the search results from the data source. In this task, we will fetch the search results using the data source method we just added above, and then bind the results data to the page.

1. Open the SearchResultsPage.xaml file and take a close look at the structure of the page. You will notice use of the **filtersItemsControl** and the **resultsGridView** properties. **ItemsControl** is used to show the search filters while **GridView** is used to display the search results.

NOTE ▶ The default page template for SearchResultsPage.xaml already has significant code for XAML and for the code-behind class file. You will notice that the <Page.Resources> within the XAML defines two instances of **CollectionViewSource** (To read more about the **CollectionViewSource** class, refer to the [MSDN article](http://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.data.collectionviewsource.aspx)). These instances are bound to **Results** and **Filters** properties of the **DefaultViewModel** for the page. You will also notice significant code in the code-behind class file regarding setting the filters and handling the filter-check event.

1. Open the code-behind file SearchResultsPage.xaml.cs and take a close look at the **navigationHelper\_LoadState** method. This method is responsible for fetching the search results and for setting up the necessary search filters.
2. Add the following private variables in the class; these will be used throughout the class.

|  |
| --- |
| C# |
| * + 1. private List<Product> \_searchResults;     2. private string \_queryText; |

Code Sample 18. Add private class members

1. Update the **navigationHelper\_LoadState** method and add the code to fetch the search results from the products data source as well as the code to set up the filters. Refer to the code sample below.

NOTE ▶ Since the method for searching products is asynchronous within the data source, you will need to use **await** while calling the **SearchProductsAsync** method from the **navigationHelper\_LoadState** method within the page. To use **await**, you must make **navigationHelper\_LoadState** asynchronous.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// Populates the page with content passed during navigation. Any saved state is also     3. /// provided when recreating a page from a prior session.     4. /// </summary>     5. /// <param name="navigationParameter">The parameter value passed to     6. /// <see cref="Frame.Navigate(Type, Object)"/> when this page was initially requested.     7. /// </param>     8. /// <param name="pageState">A dictionary of state preserved by this page during an earlier     9. /// session. This will be null the first time a page is visited.</param>     10. private async void navigationHelper\_LoadState(object sender, LoadStateEventArgs e)     11. {     12. \_queryText = e.NavigationParameter as String;     13. // Fetch the search results from ProductDataSource     14. \_searchResults = await ProductDataSource.SearchProductsAsync(\_queryText);     15. // Add the filters for each category of products     16. var filterList = new List<Filter>();     17. filterList.Add(new Filter("All", \_searchResults.Count, true));     18. var categories = await CategoryDataSource.GetListAsync();     19. foreach (var category in categories)     20. {     21. var products = (\_searchResults.Where(item => item.Category.Equals(category.Id)));     22. if (products.Count() > 0) filterList.Add(new Filter(category.Name, products.Count(), false));     23. }     25. // Communicate results through the view model     26. this.DefaultViewModel["QueryText"] = " " + '\u201c' + \_queryText + '\u201d';     27. this.DefaultViewModel["Filters"] = filterList;     28. this.DefaultViewModel["ShowFilters"] = filterList.Count > 1;     29. this.DefaultViewModel["Results"] = \_searchResults;     30. } |

Code Sample 19. Load state method for search results page

1. Add the required namespace in the file SearchResultsPage.xaml.cs.

|  |
| --- |
| C# |
| * + 1. using ContosoElectronics.DataModel;     2. using ContosoElectronics.DataSource; |

Code Sample 20. Add required namespace

1. Build and run the application. You will notice that no products details are shown within the page as the related data template is not applied yet.

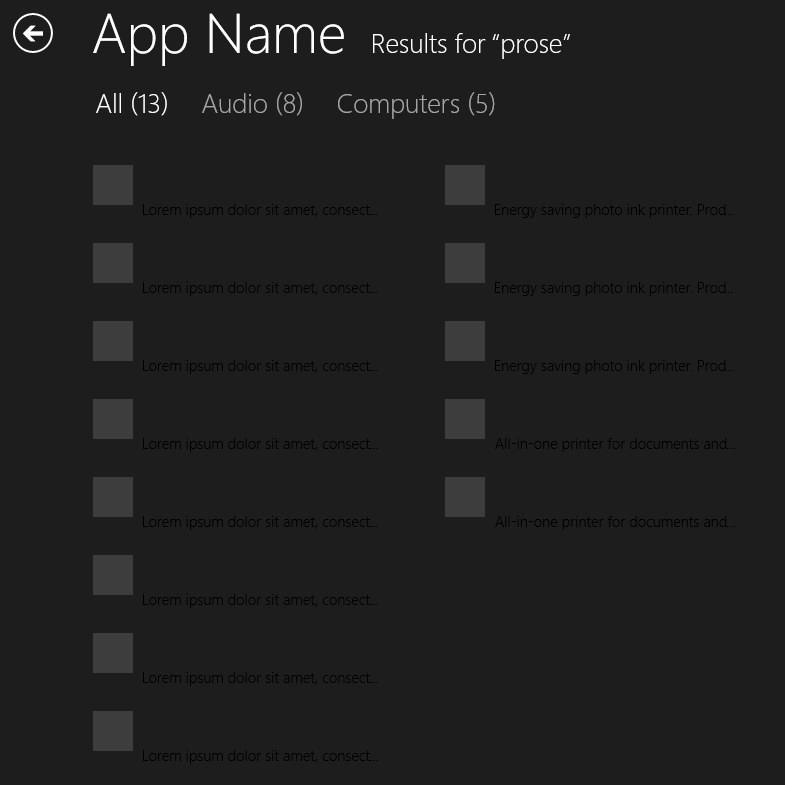


Figure 15. Search results page

1. The next step is to apply the correct data template to the search results. Open the file SearchResultsPage.xaml and replace the **ItemTemplate** assignment with the following line of code.

|  |
| --- |
| XAML |
| * + 1. ItemTemplate="{StaticResource ProductItemTemplate}" |

Code Sample 21. Assign the correct template

1. Change the height property of **ItemContainerStyle** to 100 instead of the default 70 so that the template fits correctly.

|  |
| --- |
| XAML |
| * + 1. <GridView.ItemContainerStyle>     2. <Style TargetType="Control">     3. <Setter Property="Height" Value="110"/>     4. <Setter Property="Margin" Value="0,0,38,8"/>     5. </Style>     6. </GridView.ItemContainerStyle> |

Code Sample 22. Change the item container height

1. Build and run the application. You will notice that the products items are now visible within the search results page as shown in the figure below.

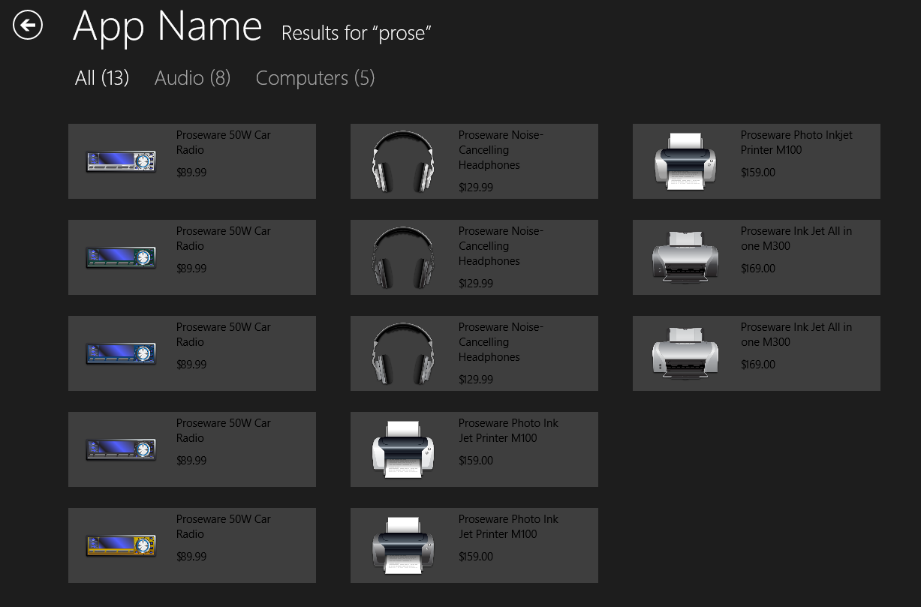


Figure 16. Search results page with the proper template

1. At this point, the filters still won’t work (try clicking one of them). To handle the filter change event, go to the SearchResultsPage.xaml.cs file and find the **Filter\_Checked** method. Change the method by adding **Async** to the method signature and reloading the search results based on the current filter, as shown below.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// Invoked when a filter is selected using a RadioButton when not snapped.     3. /// </summary>     4. /// <param name="sender">The selected RadioButton instance.</param>     5. /// <param name="e">Event data describing how the RadioButton was selected.</param>     6. async void Filter\_Checked(object sender, RoutedEventArgs e)     7. {     8. var filter = (sender as FrameworkElement).DataContext;     9. // Mirror the change into the CollectionViewSource.     10. // This is most likely not needed.     11. if (filtersViewSource.View != null)     12. {     13. filtersViewSource.View.MoveCurrentTo(filter);     14. }     15. // Determine what filter was selected     16. var selectedFilter = filter as Filter;     17. if (selectedFilter != null)     18. {     19. // Mirror the results into the corresponding Filter object to allow the     20. // RadioButton representation used when not snapped to reflect the change     21. selectedFilter.Active = true;     22. // Load the search results and filter them based on user-selected filter value     23. \_searchResults = await ProductDataSource.SearchProductsAsync(\_queryText);     24. ProductCategory category = await CategoryDataSource.GetCategoryDetailsAsync(selectedFilter.Name);     25. \_searchResults = (selectedFilter.Name.Equals("All")) ? \_searchResults :     26. new List<Product>(\_searchResults.Where(item => item.Category.Equals(category.Id)));   this.DefaultViewModel["Results"] = \_searchResults;   * + 1. // Ensure results are found     2. object results;     3. ICollection resultsCollection;     4. if (this.DefaultViewModel.TryGetValue("Results", out results) &&     5. (resultsCollection = results as ICollection) != null &&     6. resultsCollection.Count != 0)     7. {     8. VisualStateManager.GoToState(this, "ResultsFound", true);     9. return;     10. }     11. }     12. // Display informational text when there are no search results.     13. VisualStateManager.GoToState(this, "NoResultsFound", true);     14. } |

Code Sample 23. Filter check event handler

1. Build and re-run the application. Perform any search and then click on any of the filter. You would notice that only specific number of filtered results are shown for each filter.
2. When a product item is clicked, the details page for that product should open. To enable this, add the following click event in the **GridView** definition of the SearchResultsPage.xaml file.

|  |
| --- |
| XAML |
| * + 1. ItemClick="ItemGridView\_OnItemClick" |

Code Sample 24. Item click event

1. The event handler also has to be defined in SearchResultsPage.xaml.cs to make it possible to navigate to the product details page; the code for this is below.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method handles the OnItemClick event of the ItemGridView control.     3. /// On clicking a product, the user is navigated to the details page for that product.     4. /// The Product ID is passed as a parameter.     5. /// </summary>     6. private void ItemGridView\_OnItemClick(object sender, ItemClickEventArgs e)     7. {     8. this.Frame.Navigate(typeof(ProductDetailsPage), (e.ClickedItem as Product).Id);     9. } |

Code Sample 25. Item click event handler

### Task 5: Update Search Results Page Styles

Since this is a new page that we’ve added to the project, the proper styles haven’t yet been applied to it. Our final task in this exercise is to implement a style for the search results page that is consistent with the rest of the application.

1. Make the following changes to the parent grid. The idea is to use a consistent style that has already been defined in CustomStyles.xaml. Also, the height of the header row has to be reduced to 120, as shown below.

|  |
| --- |
| XAML |
| * + 1. <Grid Style="{StaticResource LayoutRootStyle}">     2. <Grid.RowDefinitions>     3. <RowDefinition Height="120"/>     4. <RowDefinition Height="\*"/>     5. </Grid.RowDefinitions>     6. …     7. </Grid> |

Code Sample 26. Define the styles for the parent grid

1. For the page title and back button styles, replace the existing code with the code below.

|  |
| --- |
| XAML |
| * + 1. <!-- Back button and page title -->     2. <Grid Background="{StaticResource PageHeaderBackgroundBrush}">     3. <Grid.ColumnDefinitions>     4. <ColumnDefinition Width="120"/>     5. <ColumnDefinition Width="Auto" />     6. <ColumnDefinition Width="Auto" />     7. <ColumnDefinition Width="\*"/>     8. </Grid.ColumnDefinitions>     9. <AppBarButton x:Name="backButton" Icon="Back" Height="95" Margin="10,28,10,0"     10. Command="{Binding NavigationHelper.GoBackCommand, ElementName=pageRoot}"     11. Visibility="{Binding IsEnabled, Converter={StaticResource BooleanToVisibilityConverter}, RelativeSource={RelativeSource Mode=Self}}"     12. AutomationProperties.Name="Back" AutomationProperties.AutomationId="BackButton"     13. AutomationProperties.ItemType="Navigation Button"/>     14. <TextBlock x:Name="pageTitle" Text="{StaticResource AppName}" Style="{StaticResource HeaderTextBlockStyle}" Grid.Column="1" IsHitTestVisible="false" TextWrapping="NoWrap" VerticalAlignment="Bottom" Margin="0,0,30,40"/>     15. <TextBlock x:Name="resultText" Grid.Column="2" Text="Results for&#x00a0; " IsHitTestVisible="false" Style="{StaticResource SubheaderTextBlockStyle}" TextWrapping="NoWrap" VerticalAlignment="Bottom" Margin="0,0,0,40" />     16. <TextBlock x:Name="queryText" Grid.Column="3" Text="{Binding QueryText}" IsHitTestVisible="false" Style="{StaticResource SubheaderTextBlockStyle}" TextWrapping="NoWrap" VerticalAlignment="Bottom" Margin="0,0,0,40" />     17. </Grid> |

Code Sample 27. Applying page header styles

1. Now, to make it more appealing, we need to apply a minor adjustment to the filter list control. Replace the existing code for **ItemsControl** with the code provided below.

|  |
| --- |
| XAML |
| <ItemsControl  x:Name="filtersItemsControl"  Canvas.ZIndex="1"  Margin="116,20,120,30"  ItemsSource="{Binding Source={StaticResource filtersViewSource}}"  Visibility="{Binding ShowFilters, Converter={StaticResource BooleanToVisibilityConverter}}">  <ItemsControl.ItemsPanel>  <ItemsPanelTemplate>  <StackPanel Orientation="Horizontal"/>  </ItemsPanelTemplate>  </ItemsControl.ItemsPanel>  <ItemsControl.ItemTemplate>  <DataTemplate>  <RadioButton  Margin="0,0,30,0"  GroupName="Filters"  IsChecked="{Binding Active, Mode=TwoWay}"  Checked="Filter\_Checked"  Style="{StaticResource TextBlockButtonStyle}">  <TextBlock Text="{Binding Description}" TextWrapping="NoWrap"  Margin="3,-7,3,10" Style="{StaticResource SubheaderTextBlockStyle}"  Foreground="{StaticResource ApplicationForegroundThemeBrush}"/>  </RadioButton>  </DataTemplate>  </ItemsControl.ItemTemplate>   * + 1. </ItemsControl> |

Code Sample 28. Applying styles to the items control

1. Change the value of the **AppName** string to “Search Results” to give it an appropriate name.

|  |
| --- |
| XAML |
| * + 1. <x:String x:Key="AppName">Search Results</x:String> |

Code Sample 29. Name the page

1. Build and run the application. Evoke the Search charm and conduct a search. The final screen should appear as shown in the figure below.

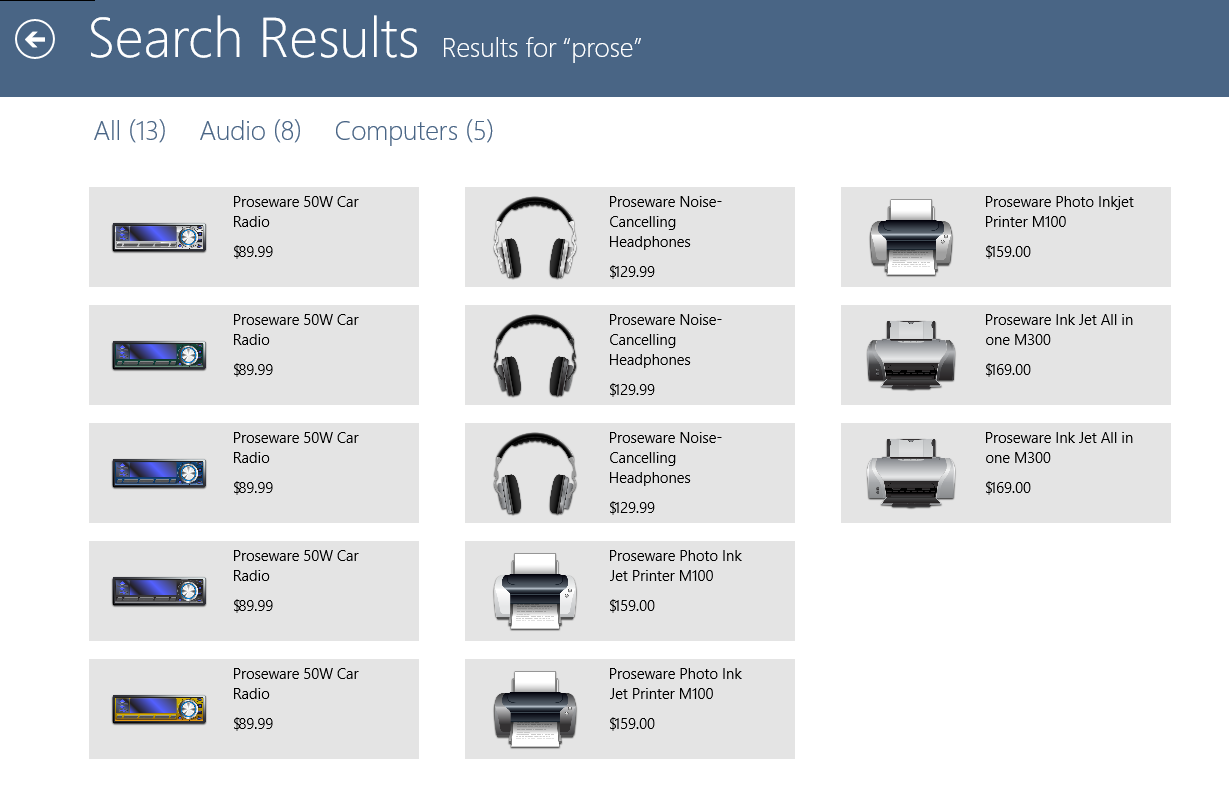


Figure 17. The completed search results page

## Exercise 4: Share

Like search, Windows Store apps expose share in a consistent fashion. Swiping on the right edge of the screen or pressing the Windows+C key combination invokes the charm bar. You can implement share as a default function for the app as a whole and/or customize it for specific content within the app. In the default mode of sharing, a screenshot of the current page is taken, which you can then email. Sharing specific content on a page requires customization of the code for the Share contract.

In this exercise you will learn how to implement the Share contract for the product details page of the Contoso Electronics application. You will enable the product details page to share information about the product—product name, specifications, price—in a structured manner. The rest of the pages in the application will use default sharing behavior.

There are two tasks in this exercise:

1. Add handler for Share charm invocation.
2. Define the content structure to share.

### Task 1: Add Handler for Share Charm Invocation

Before we start this task, build and run the application, and then invoke the Share charm for any page. You will notice that the page shows the default behavior for sharing, capturing a screenshot of the page to be shared.

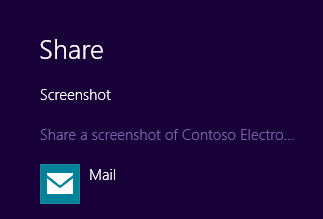


Figure 18. Default Share charm behavior

On selecting **Mail**, a sample email is created with the screenshot of the shared page as an attachment.

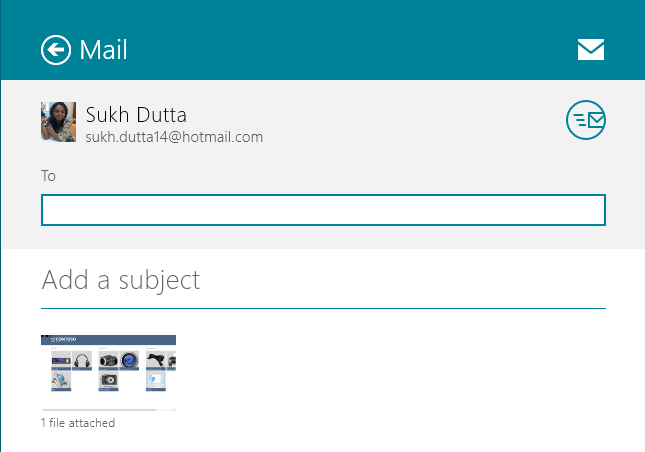


Figure 19. Sample email with a screen capture of the current page as an attachment

Now, to customize share functionality for the product details page, the first task is to create an object of the **DataTransferManager,** which is the starting point of any sharing operation and defines the handler that handles the **DataRequested** event. The [**DataRequested**](http://msdn.microsoft.com/en-us/library/windows/apps/windows.applicationmodel.datatransfer.datatransfermanager.datarequested.aspx) event occurs when the user taps or clicks the Share charm.

1. First you need to add the right namespaces to your app so you can create and process the objects related to sharing. Add the following namespace to the ProductDetailsPage.xaml.cs page.

|  |
| --- |
| C# |
| * + 1. using Windows.ApplicationModel.DataTransfer; |

Code Sample 30. Add namespace to ProductDetailsPage.xaml.cs

1. The next step is to get an instance of the **DataTransferManager** and setup the handlers for the **DataRequested** event. In the ProductDetailsPage.xaml.cs page, update the **OnNavigatedTo** and the **OnNavigatedFrom** methods as shown below.

|  |
| --- |
| C# |
| * + 1. protected override void OnNavigatedTo(NavigationEventArgs e)  {      //Call the corresponding method on Navigation helper      navigationHelper.OnNavigatedTo(e);     2. //Get an instance of DataTransferManager and add handler for DataRequested event      DataTransferManager dataTransferManager = DataTransferManager.GetForCurrentView();      dataTransferManager.DataRequested += DataRequested;             } |

Code Sample 31. Add Handler for DataTransferManager.DataRequested event

|  |
| --- |
| C# |
| * + 1. protected override void OnNavigatedFrom(NavigationEventArgs e)  {      //Call the corresponding method on Navigation helper      navigationHelper.OnNavigatedFrom(e);     2. //Get an instance of DataTransferManager and remove handler for DataRequested event      DataTransferManager dataTransferManager = DataTransferManager.GetForCurrentView();      dataTransferManager.DataRequested -= DataRequested;             } |

Code Sample 32. Remove Handler for DataTransferManager.DataRequested event

1. Now, add the **DataRequested** handler to the page by adding the requisite code to the ProductDetailsPage.xaml.cs file, as shown below.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method contains the logic to prepare the data which would be shared. The product     3. /// details would be prepared in the form of a table which can then be shared through email.     4. /// </summary>     5. private void DataRequested(DataTransferManager sender, DataRequestedEventArgs args)     6. {     7. } |

Code Sample 33. Add DataRequested handler

1. Build and run the application, and then evoke the Share charm for the product details page. You will notice that the page no longer shows the default sharing behavior. Instead it offers two options: the default involving a screenshot, and another labeled “Contoso Electronics,” as shown below.

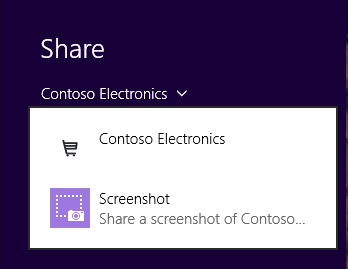


Figure 20. Sharing options for product details page

1. Select **Contoso Electronics**. As the handler we created has no code, you should see the message “There’s nothing to share right now,” shown below.

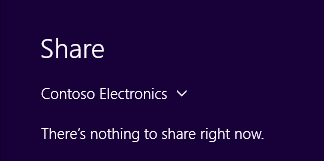


Figure 21. Intermediate state of share without the content defined

### Task 2: Define the Content Structure to Share

Let’s solve this problem now by defining the actual content to be shared from the product details page.

1. Add the following code to the **DataRequested** handler. This transforms the content into HTML for sharing.

|  |
| --- |
| C# |
| * + 1. DataRequest request = args.Request;     2. if (request != null)     3. {     4. request.Data.Properties.Title = "Found an interesting product...";     5. request.Data.Properties.Description = product.Description;     6. //Create table format using the Product details     7. StringBuilder htmlcontent = new StringBuilder("<p> <Table style=\"font-family:Segoe UI;\"");     8. htmlcontent.Append("<tr> <td> <b> Product Name: </b>" + product.Name + " </td> </tr> ");     9. htmlcontent.Append("<tr> <td> <b> Product Number: </b>" + product.Id + " </td> </tr> ");     10. htmlcontent.Append("<tr> <td> <b> Product Price: </b>$" + PriceConverter.Convert(product.Price) + " </td> </tr> ");     11. htmlcontent.Append("<tr> <td> </td> </tr> <tr> <td> <b> Description: </b> </td> </tr> ");     12. htmlcontent.Append("<tr> <td> " + product.Description + " </td></tr> ");     13. if (product.ProductSpecifications != null)     14. {     15. htmlcontent.Append("<tr> <td> </td> </tr> <tr> <td> <b> Specifications: </b> </td> </tr> ");     16. foreach (var spec in product.ProductSpecifications)     17. {     18. htmlcontent.Append("<tr> <td> " + spec.Name + ": " + spec.Value + " </td></tr> ");     19. }     20. }     21. htmlcontent.Append("</table> </p>");     22. request.Data.SetHtmlFormat(HtmlFormatHelper.CreateHtmlFormat(htmlcontent.ToString()));     23. } |

Code Sample 34. Create content in HTML format for sharing

NOTE ▶ In addition to HTML, you can share content through the Share contract using a variety of formats. Here, we have chosen HTML in order to show the data in a simple table format. For more information about choosing the appropriate data format per your requirements, refer to [the MSDN article](http://msdn.microsoft.com/en-us/library/windows/apps/hh871364.aspx).

1. Build and run the application and then evoke the Share charm for the product details page. You will notice that the two sharing options are now updated: you can either share a screenshot of the page or share the product details for the current product, as shown immediately below.

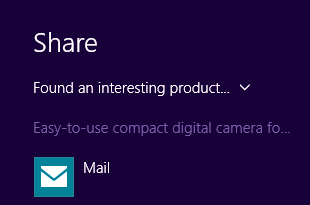


Figure 22. New options for sharing product details page after content definition

1. Click the **Found an interesting product…** option. The content to be shared in now neatly formatted and added in a sample e-mail, as shown below.



Figure 23. Sample e-mail with the content defined

## Exercise 5: App Bar

App bars are used to present secondary navigation, commands, and tools that can be hidden away when they aren't needed. You can put an app bar at the top of the page, at the bottom of the page, or both. App bars are hidden by default, and are shown or dismissed when the user right-clicks on a mouse, presses the Windows+Z key combination, or swipes from the top or bottom edge of the screen. They can also be shown programmatically when the user makes a selection or interacts with the app.

In this exercise we are going to add the app bar to the bottom of the product details page, and then put a navigational command button on it.

NOTE ▶ For more information on app bar controls in Windows 8.1, see the [MSDN Web site](http://msdn.microsoft.com/en-us/library/windows/apps/bg182878.aspx#AppBarControls). For detailed guidance on how to implement them optimally, refer to [the MSDN article](http://msdn.microsoft.com/en-us/library/windows/apps/hh465302.aspx).

### Task 1: Add Bottom App Bar

1. Open the ProductDetailsPage.xaml file.
2. Append the following piece of XAML code inside **Page** tag.

|  |
| --- |
| XAML |
| * + 1. <Page.BottomAppBar>     2. <CommandBar>     4. </CommandBar>     5. </Page.BottomAppBar> |

Code Sample 35. Adding an app bar in ProductDetailsPage.xaml

### Task 2: Add Command to App Bar

The app bar in place, now the task is to add a command to it. For this exercise, we’ll add a navigation command button that takes the user back to the group categories page. In Windows 8.1, app bar commands are classified as either primary or secondary, depending on the importance of the command in the context of a given scenario. Since navigating to the group categories page is a secondary action for the page, we will add it to the **SecondaryCommands** collection.

1. Open the ProductDetailsPage.xaml file.
2. Add the following code within the **CommandBar** tags.

|  |
| --- |
| XAML |
| * + 1. <Page.BottomAppBar>     2. <CommandBar>     3. <CommandBar.SecondaryCommands>     4. <AppBarButton x:Name="Home" Label="Home" Icon="Home" Click="Home\_Click"/>     5. </CommandBar.SecondaryCommands>     6. </CommandBar>   </Page.BottomAppBar> |

Code Sample 36. Add command button to the app bar

1. Open ProductDetailsPage.xaml.cs and add the following code, which contains the event handler for the app bar button.

|  |
| --- |
| XAML |
| * + 1. /// <summary>     2. /// Handles the App Bar event for Home command.     3. /// </summary>     4. private void Home\_Click(object sender, RoutedEventArgs e)     5. {     6. this.Frame.Navigate(typeof(CategoryListingPage));     7. } |

Code Sample 37. Add event handler for app bar button

1. Run the application and navigate to the customer details page.
2. Right-click the screen using your mouse (or swiping from top or bottom if you’re on a touchscreen). The app bar will be summoned from the bottom of the page, as shown below.



Figure 24. App bar with a navigational command button

## Exercise 6: Live Tiles

A tile is used to represent an app on the Start screen, and can be configured to present rich and engaging content to users when the app is not running. Tapping or clicking the tile launches the app.

Tiles can be live (updated through notifications) or you can leave them static. Tiles begin in a default state, defined in the app's manifest. A static tile will always display the default content, which is generally a full-tile logo image. A live tile can update the default tile to show new content, but can return to the default if the update expires or is removed. A tile can also display a status badge, which can be a number—such as the quantity of emails or messages in queue—or a glyph.

In this exercise, you will learn how to implement a live tile for the Contoso Electronics application by completing three tasks:

1. Choose the type of live tile.
2. Configuring the content for the live tile.
3. Enable the live tile.

### Task 1: Choose the Type of Live Tile

A Windows 8.1-based Windows Store app has four kinds of tiles (small, wide, medium, large), and of these, all except the small variety can be configured for live updates. Visual Studio 2013 also provides many formats for each kind of live tile, giving you many options for configuring your tile to look and behave the way you want.

For this lab, we will configure the live tile functionality for a wide tile. Among the wide tile formats, we want to show the content as two lines of text: with product price on the first line, product name on the second, and a product image on the left side of the tile.

1. Open the App.xaml.cs file.
2. Add the method shown below.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method takes a Product as its input and generates an XML Document which contains actual     3. /// values to be displayed in the live tile.   /// </summary>   * + 1. private XmlDocument CreateWideTile(Product product)     2. {     3. // Create a live update for a wide tile     4. XmlDocument wideTileXml = TileUpdateManager.GetTemplateContent(TileTemplateType.TileWide310x150SmallImageAndText04);     5. // Assign text     6. XmlNodeList wideTileTextAttributes = wideTileXml.GetElementsByTagName("text");     7. wideTileTextAttributes[0].AppendChild(wideTileXml.CreateTextNode(product.Price.ToString("C")));     8. wideTileTextAttributes[1].AppendChild(wideTileXml.CreateTextNode(product.Name));     9. // Assign Image     10. XmlNodeList wideTileImageAttributes = wideTileXml.GetElementsByTagName("image");     11. ((XmlElement)wideTileImageAttributes[0]).SetAttribute("src", product.ImagePath);     12. ((XmlElement)wideTileImageAttributes[0]).SetAttribute("alt", "Contoso Electronics logo");     13. return wideTileXml;     14. } |

Code sample 38. Generate XML document for values to be displayed in live tile

### Task 2: Configure the Content for the Live Tile

Now that we have decided how our live tile should look, we need to fetch the data to be displayed in the live tile and get the XML schema for the data.

1. Add the following method in ProductDataSource.cs; this will return a selection of random products for the live tile to cycle through and display.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// Gets some random products for live tile to flip through.     3. /// </summary>     4. /// <param name="count">Number of random products</param>     5. /// <returns>List of products</returns>     6. public static async Task<List<Product>> GetRandomProductsAsync(int count)     7. {     8. await \_productDataSource.ReadXmlDataFromLocalStorageAsync();     9. var matches = \_productDataSource.AllProducts.OrderBy(item => Guid.NewGuid()).Take(count).ToList();     10. return matches;     11. } |

Code Sample 39. Add method for fetching random products

1. In the App.xaml.cs file, add the following code to fetch the list of random products, cycle through them, and create the XML document for each product.

|  |
| --- |
| C# |
| * + 1. /// <summary>   /// This method creates a new notification for each product by cycling through a list of  /// random products and passes the notification to the TileNotificationManager.   * + 1. /// It then enables notification for the application.     2. /// </summary>     3. private async void CreateLiveTiles()     4. {     5. var randomProducts = await ProductDataSource.GetRandomProductsAsync(3);     6. foreach (var product in randomProducts)     7. {     8. // Create a tile notification.     9. TileNotification tileNotification = new TileNotification(CreateWideTile(product));     10. TileUpdateManager.CreateTileUpdaterForApplication().Update(tileNotification);     11. }     12. } |

Code Sample 40. Create an XML document for each product

### Task 3: Enable the Live Tile

1. Add the following code at the end of the **OnLaunched** method of the App.xaml.cs file.

|  |
| --- |
| C# |
| * + 1. // Call the method to create Live tiles     2. CreateLiveTiles(); |

Code Sample 41. Call the method to create live tiles

1. Add the following code at the end of the **CreateLiveTiles** method of the App.xaml.cs file. This line of code enables the tile to queue up to five notifications.

|  |
| --- |
| C# |
| * + 1. TileUpdateManager.CreateTileUpdaterForApplication().EnableNotificationQueue(true); |

Code Sample 42. Enable tile to queue notifications

1. Build and run the application and then navigate to your Windows Start screen. You should be able to see the Contoso Electronics wide tile cycling through the list of random products, as shown below.



Figure 25. Live tile for the Contoso Electronics app

# Optional Exercises

## Exercise 1: Semantic Zoom

Semantic Zoom is a touch-optimized technique used by Windows Store apps for presenting and navigating large sets of related data or content within a single view. Semantic Zoom uses two distinct modes of classification (or zoom levels) for organizing and presenting the content: one low-level (or zoomed-in) mode that is typically used to display items in a flat, all-up structure and another, high-level (or zoomed-out) mode that displays items in groups and enables a user to quickly navigate and browse through the content.

This exercise will enable you to create a zoomed out view of the list of product categories that shows the product categories represented by tiles and the number of subcategories within each.

This exercise involves the following 2 tasks:

1. Add Semantic Zoom control.
2. Define the **ZoomedOutView**.

### Task 1: Add Semantic Zoom control

This task will enable you to define a Semantic Zoom control with your original **GridView** as the zoomed-in section.

1. Open the CategoryListingPage.xaml file and add the following lines of code.

|  |
| --- |
| XAML |
| * + 1. <SemanticZoom Grid.RowSpan="2" x:Name="SemanticZoom">     2. <SemanticZoom.ZoomedInView>     3. </SemanticZoom.ZoomedInView>     4. </SemanticZoom> |

Code Sample 43. Add the Semantic Zoom control

1. Move **itemGridView** inside the **SemanticZoom.ZoomedInView** tag.

|  |
| --- |
| XAML |
| * + 1. <SemanticZoom Grid.RowSpan="2" x:Name="SemanticZoom">     2. <SemanticZoom.ZoomedInView>     3. <!-- Horizontal Scrolling GridView -->     4. <GridView     5. x:Name="itemGridView"     6. AutomationProperties.AutomationId="ItemGridView"     7. AutomationProperties.Name="Grouped Items"     8. …     9. </GridView>     10. </SemanticZoom.ZoomedInView>   </SemanticZoom> |

Code Sample 44. Move GridView inside ZoomedInView

### Task 2: Define the ZoomedOutView

For **ZoomedOutView**, we will be showing a high level list of the group names along with the number of items in each group.

1. Check if the style **ProductCategoryItemTemplate** style is present in CustomTemplates.xaml file. If it is not, add the following code to the CustomTemplates.xaml template.

|  |
| --- |
| XAML |
| * + 1. <DataTemplate x:Key="ProductCategoryItemTemplate">     2. <Grid Height="280" Width="240" >     3. <Grid.Background>     4. <SolidColorBrush Color="{StaticResource TileColor}" Opacity="0.5"/>     5. </Grid.Background>     6. <TextBlock Text="{Binding Group.SubCategoryItems.Count}" VerticalAlignment="Top" TextAlignment="Right" TextWrapping="NoWrap" Margin="0,0,20,0" FontSize="104" FontWeight="Light" Foreground="{StaticResource ApplicationForegroundThemeBrush}" />     7. <TextBlock Text="{Binding Group.Name}" TextAlignment="Left" Foreground="{StaticResource ApplicationForegroundThemeBrush}" VerticalAlignment="Bottom" Margin="15" Style="{StaticResource SubheaderTextBlockStyle}"/>     8. </Grid>     9. </DataTemplate> |

Code Sample 45. Item template for zoomed-out GridView items

1. Add the following XAML code to your page GroupedCategoriesPage.xaml after the **SemanticZoom.ZoomedOutView** closing tag. This is for creating the zoomed-out view.

|  |
| --- |
| XAML |
| * + 1. <SemanticZoom.ZoomedOutView>     2. <GridView x:Name="ZoomedOutGrid"     3. VerticalAlignment="Center"     4. ItemTemplate="{StaticResource ProductCategoryItemTemplate}" >     5. <GridView.ItemsPanel>     6. <ItemsPanelTemplate>     7. <StackPanel Orientation="Horizontal" Margin="116,136,40,46"/>     8. </ItemsPanelTemplate>     9. </GridView.ItemsPanel>     10. </GridView>     11. </SemanticZoom.ZoomedOutView> |

Code Sample 46. Add zoomed-out view

1. The last step is to assign the data to the **ItemsSource** of the zoomed-out **GridView**. This uses the same data as the zoomed-in view but in a more grouped manner. Copy the following code into the **LoadState** method of the GroupedCategoriesPage.xaml.cs file.

|  |
| --- |
| C# |
| * + 1. // Assign the groups as the data source for Zoomed Out view     2. (SemanticZoom.ZoomedOutView as ListViewBase).ItemsSource = groupedItemsViewSource.View.CollectionGroups; |

Code Sample 47. Assign data source to the zoomed-out view

1. Build and run the application. You should see a minus symbol (“-“) at the bottom right corner of the screen. Click it if you are using a mouse, or use a pinch gesture in a touchscreen environment. It should now be zoomed-out, as shown below.



Figure 26. Zoomed-out view in the Contoso Electronics app

1. Click any of the tiles. The screen should zoom in to the subcategories of the given tile’s category.

## Exercise 2: Windowing Modes

In Windows 8.1, there is no fixed-width view state. Instead, users can now resize apps continuously down to a minimum width. (The default minimum width of an app is 500 pixels.) So apps no longer have the snapped and filled view states. Instead, you develop your app to be functional and good-looking at any size down to the minimum.

This exercise will show you how to create a narrower view for the grouped categories page, which you can resize down to a defined minimum width.

This exercise includes four tasks:

1. Define the minimum width for the application.
2. Define the narrower view for the page.
3. Set up the visual states.
4. Handle the **Window.SizeChanged** event.

### Task 1: Define the Minimum Width for the Application

In this task, you will define the minimum width to which the window can be resized.

1. Open the Package.appxmanifest file from the Solution Explorer and then select the **Application UI** tab.
2. Find the **Minimum width** box, which is not set by default. Change the option to 320 px. Now your application can be narrowed down to 320 px.

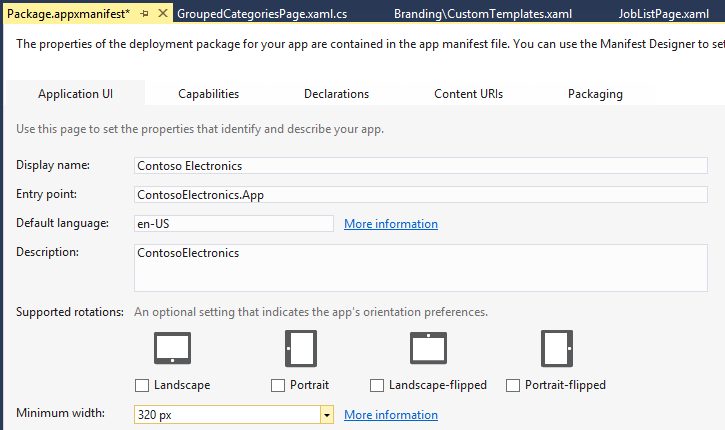


Figure 27. Changing the minimum window width in the manifest

### Task 2: Define the Narrower View for the Page

Consider the structure and layout of the content beyond a specific width. It may make more sense to display the list of product subcategories as a vertical scrolling list when the width of the page becomes narrower than 620 px.

1. Add the following XAML code to CategoryListingPage.xaml inside the main layout grid. This is to define the structure of a **ListView** control that will hold the subcategories when the page is narrowed.

|  |
| --- |
| XAML |
| * + 1. <!--     2. ======================================================================     3. DEFINE SECONDARY VIEW:     5. This is the narrow view for the page. This view is used when the page     6. is resized beyond a specific width. This view represents the vertical     7. listing of the sub-category items using ListView.     8. ======================================================================     9. -->     10. <ListView     11. x:Name="itemListView"     12. AutomationProperties.AutomationId="ItemListView"     13. AutomationProperties.Name="Grouped Items"     14. Grid.Row="1"     15. Visibility="Collapsed"     16. ItemsSource="{Binding Source={StaticResource groupedItemsViewSource}}"     17. ItemTemplate="{StaticResource SnappedSubCategoryTemplate}"     18. SelectionMode="None"     19. IsSwipeEnabled="false"     20. IsItemClickEnabled="True"     21. ItemClick="ItemGridView\_OnItemClick">     22. <ListView.ItemsPanel>     23. <ItemsPanelTemplate>     24. <StackPanel Orientation="Vertical" Margin="16,20,16,40"/>     25. </ItemsPanelTemplate>     26. </ListView.ItemsPanel>     27. <ListView.ItemContainerStyle>     28. <Style TargetType="ListViewItem">     29. <Setter Property="Margin" Value="0" />     30. </Style>     31. </ListView.ItemContainerStyle>     32. <ListView.GroupStyle>     33. <GroupStyle>     34. <GroupStyle.HeaderTemplate>     35. <DataTemplate>     36. <TextBlock Text="{Binding Title}" Margin="4,0,0,10" Style="{StaticResource SubheaderTextBlockStyle}" />     37. </DataTemplate>     38. </GroupStyle.HeaderTemplate>     39. <GroupStyle.Panel>     40. <ItemsPanelTemplate>     41. <VariableSizedWrapGrid Orientation="Horizontal" Margin="0,0,0,10"/>     42. </ItemsPanelTemplate>     43. </GroupStyle.Panel>     44. </GroupStyle>     45. </ListView.GroupStyle>     46. </ListView> |

Code Sample 48. ListView control configured for a narrow, "snapped" view

### Task 3: Set Up the Visual States

Now, the task is to handle the various visual states (normal, narrow, etc.) and define how the visual elements on your page will appear in each state.

1. Check to see how the page responds on resizing. The horizontal scrolling **GridView** should be invisible when the application is resized below 620 px. At this width, the **ListView** that you added in the previous task should become visible.
2. Add the following XAML code to your page, CategoryListingPage.xaml, inside the main layout grid. This is for handling the visual states.

|  |
| --- |
| XAML |
| * + 1. <!-- ===================================== -->     2. <!-- VISUAL STYLES -->     3. <!-- This is the narrow view for the page. -->     4. <!-- ===================================== -->     5. <VisualStateManager.VisualStateGroups>     6. <VisualStateGroup x:Name="LayoutStates">     7. <VisualState x:Name="PrimaryLayout">     8. <Storyboard></Storyboard>     9. </VisualState>     10. <VisualState x:Name="NarrowLayout">     11. <Storyboard>     12. <Storyboard>     13. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="pageTitle" Storyboard.TargetProperty="Margin">     14. <DiscreteObjectKeyFrame KeyTime="0" Value="-100,20,20,20"/>     15. </ObjectAnimationUsingKeyFrames>     16. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="itemListView" Storyboard.TargetProperty="Visibility">     17. <DiscreteObjectKeyFrame KeyTime="0" Value="Visible"/>     18. </ObjectAnimationUsingKeyFrames>     19. <ObjectAnimationUsingKeyFrames Storyboard.TargetName="SemanticZoom" Storyboard.TargetProperty="Visibility">     20. <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>     21. </ObjectAnimationUsingKeyFrames>     22. </Storyboard>     23. </Storyboard>     24. </VisualState>     25. </VisualStateGroup>     26. </VisualStateManager.VisualStateGroups> |

Code Sample 49. Setting up visual states for the page

### Task 4: Handle the Window.SizeChanged Event

The last task in this exercise involves handling the event that occurs whenever you change the width of the page. This event must be handled and the visual state must be selected accordingly.

1. Add the following code to the constructor in the page GroupedCategoriesPage.xaml.cs.

|  |
| --- |
| C# |
| * + 1. // Register a handler for the Window.SizeChanged event     2. Window.Current.SizeChanged += Window\_SizeChanged; |

Code Sample 50. Register a handler for the Window.SizeChanged event

1. Add the following method for changing visual states when the width of the page is made less that 620 px.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// Check the window width and update the Visual State     3. /// </summary>     4. /// <param name="width">Width of current window</param>     5. void UpdateVisualState (double width)     6. {     7. if (width < 620)     8. {     9. VisualStateManager.GoToState(this, "NarrowLayout", true);     10. }     11. else     12. {     13. VisualStateManager.GoToState(this, "PrimaryLayout", true);     14. }     15. }     16. /// <summary>     17. /// Represents the event handler for the Window.SizeChanged event.     18. /// This event is raised whenever the window is resized and helps     19. /// handle any changes to visual state of the page.     20. /// </summary>     21. void Window\_SizeChanged(object sender, WindowSizeChangedEventArgs e)     22. {   // Check the window size and update the Visual State   * + 1. UpdateVisualState(e.Size.Width);     2. } |

Code Sample 51. Method for changing visual states

1. Update the LoadState method to include the method call to **UpdateVisualState** method, as shown the in the code snippet below. This will ensure that the visual state is updated even during page load.

|  |
| --- |
| C# |
| * + 1. private async void navigationHelper\_LoadState(object sender, LoadStateEventArgs e)     2. {     3. ...     4. // Check the window size and update the Visual State     5. UpdateVisualState(Window.Current.Bounds.Width);     6. } |

Code sample 52. LoadState method with call to UpdateVisualState

1. Build and run the application. The application will be in the default view. Drag the handle and narrow the application to half the screen. It will still have the horizontal view, as shown below in the figure below.

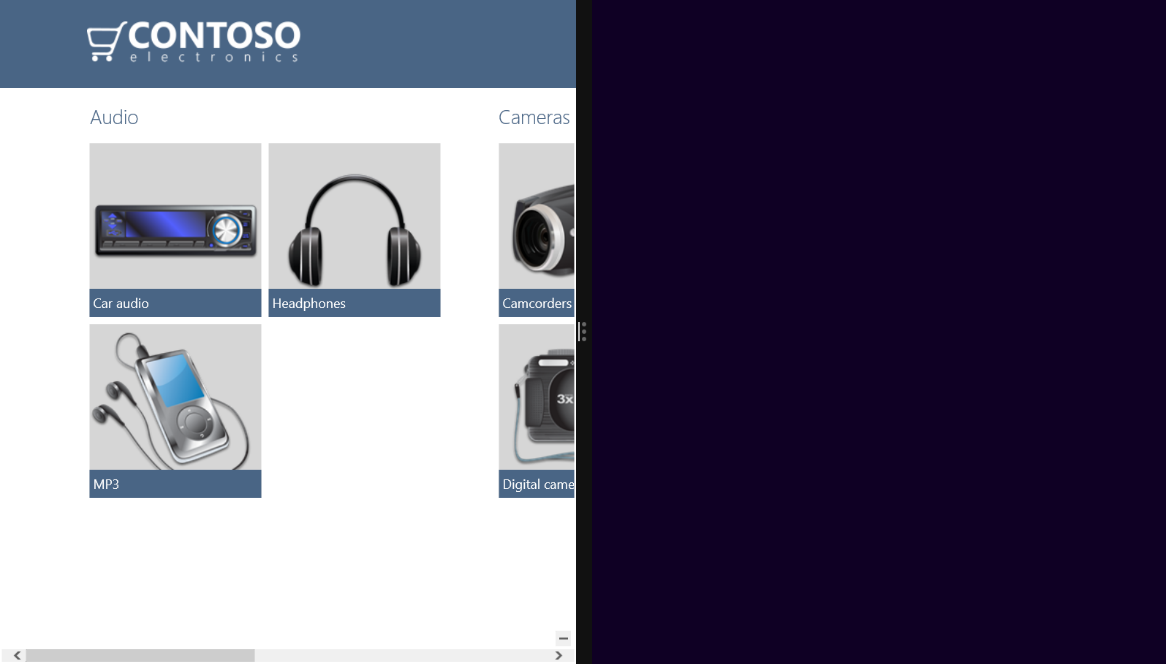


Figure 28. Default view at half screen width

1. Now decrease the width further by dragging the window handle to the left. Beyond the 620 px threshold, the view changes and becomes a vertically scrolling list of product categories and subcategories, as shown below.



Figure 29. App in a narrow view with a vertically scrolling list

## Exercise 3: Secondary Tiles

In addition to the primary tile (see above, *Exercise 6: Live Tiles*), a Windows Store app can also be configured to enable one or more secondary tiles. This features permits the pinning of specific, non-default sections of the application on the Windows Start screen so they can be accessed directly without having to go through the steps of navigating via the default entry point of the app. For example, a user may wish to pin to the Start screen a product page, article, or specific content hub of a given app for convenient reference later.

In this exercise, you will enable secondary tiles for your application so that its users may select any particular product and pin it to their Start screens. By clicking (or tapping, as the case may be) the secondary tile, your users will directly access the product details page for the pinned product.

In this exercise, there are five tasks:

1. Add the **Pin To Start** button.
2. Handle the button event.
3. Enable secondary tile pinning.
4. Enable secondary tile unpinning.
5. Enable entry through the secondary tile.

### Task 1: Add the Pin To Start Button

1. Open the ProductDetailsPage.xaml file. The page already has a bottom app bar with a **Home** button. We are now going to add a second button to the app bar, the **Pin To Start** button.
2. Append the following XAML inside the **Page** tag for the app bar we defined previously (see *Exercise 5: App Bar*).

|  |
| --- |
| XAML |
| * + 1. <Page.BottomAppBar>     2. <CommandBar>     3. <CommandBar.SecondaryCommands>     4. <AppBarButton x:Name="Home" Label="Home" Icon="Home" Click="Home\_Click"/>     5. <AppBarButton x:Name="PinToStart" Click="PinToStart\_Click" Loaded="PinToStart\_Loaded"/>     6. </CommandBar.SecondaryCommands>     7. </CommandBar>     8. </Page.BottomAppBar> |

Code Sample 53. Adding PinToStart in ProductDetailsPage.xaml

### Task 2: Handle the Button Event

The **Pin To Start** button has a click event that needs a handler.

1. Add the following code in the ProductDetailsPage.xaml.cs file.

|  |
| --- |
| C# |
| * + 1. /// <summary>   /// Handles the App Bar event - Pin To Start. This event is used to pin the current Product /// as a secondary tile on the Start screen   * + 1. /// </summary>     2. private async void PinToStart\_Click(object sender, RoutedEventArgs e)     3. {     4. await SecondaryTileCreation(sender);     5. } |

Code Sample 54. Adding the handler for the click event

1. The **SecondaryTileCreation** method checks if a secondary tile for the specific product is already present. If it is, then user is given an option to unpin the tile from the Start screen. Otherwise, the user can pin the product. Add the following code in the ProductDetailsPage.xaml.cs file.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method checks if a secondary tile is already present for this product. If it is     3. /// already present, the product is unpinned, if not the product is pinned.     4. /// </summary>     5. private async Task SecondaryTileCreation(object sender)     6. {     7. string appbarTileId = product.Id;     8. if (!SecondaryTile.Exists(appbarTileId))     9. {     10. await PinSecondaryTile(sender, appbarTileId);     11. }     12. else     13. {     14. //await UnpinSecondaryTile(sender, appbarTileId);     15. }     16. } |

Code Sample 55. Add method to check pin state for given product

### Task 3: Enable Secondary Tile Pinning

To create a secondary tile, a sample tile element (or placeholder) must be created to show the user what the secondary tile will look like when pinned.

1. Add the following code to ProductDetailsPage.xaml.cs.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method creates the placeholder for the secondary tile and shows it above the 'Pin' /// button.     3. /// </summary>     4. public static Rect GetElementRect(FrameworkElement element)     5. {     6. GeneralTransform buttonTransform = element.TransformToVisual(null);     7. Point point = buttonTransform.TransformPoint(new Point());     8. return new Rect(point, new Size(element.ActualWidth, element.ActualHeight));     9. } |

Code Sample 56. Create secondary tile placeholder

1. To fully enable pinning, you also need to make sure that the **Pin To Start** button functions as expected: once a product is pinned, the button should toggle from pin to unpin. Copy the following lines of code into the ProductDetailsPage.xaml.cs file to enable this functionality.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method assigns the style to the app bar button.     3. /// </summary>     4. private void ToggleAppBarButton(bool showPinButton, AppBarButton pinToStart)     5. {     6. if (pinToStart != null)     7. {     8. pinToStart.Icon = (showPinButton) ? new SymbolIcon(Symbol.Pin) :     9. new SymbolIcon(Symbol.UnPin);     10. pinToStart.Label = (showPinButton) ? "Pin To Start" : "Unpin";     11. }     12. } |

Code Sample 57. Enable toggling of the Pin To Start button

1. Next, we need to specify the method that actually pins a job to the Start screen. Copy the following code into the ProductDetailsPage.xaml.cs file.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method pins the secondary tile. The secondary tile is created using the required /// parameters and pinned. The user is shown a message informing whether the tile is pinned /// successfully     3. /// </summary>     4. private async Task PinSecondaryTile(object sender, string appbarTileId)     5. {     6. // Prepare package images for use as the Tile Logo in our tile to be pinned     7. Uri smallLogo = new Uri("ms-appx://" + product.ImagePath);     8. Uri wideLogo = new Uri("ms-appx:///Assets/Tile\_310X150.png");     9. string tileActivationArguments = appbarTileId;     10. // Create a 1x1 Secondary tile     11. string subTitle = product.Name;     12. SecondaryTile secondaryTile = new SecondaryTile(appbarTileId, product.Name, subTitle, tileActivationArguments, TileOptions.ShowNameOnLogo | TileOptions.ShowNameOnWideLogo, smallLogo, wideLogo);     13. secondaryTile.ForegroundText = ForegroundText.Dark;     14. bool isPinned = await secondaryTile.RequestCreateForSelectionAsync( GetElementRect((FrameworkElement)sender), Windows.UI.Popups.Placement.Above);     15. if (isPinned)     16. {     17. MessageDialog dialog = new MessageDialog("Product " + product.Name + " successfully pinned.");     18. await dialog.ShowAsync();     19. }     20. else     21. {     22. MessageDialog dialog = new MessageDialog("Product " + product.Name + " not pinned.");     23. await dialog.ShowAsync();     24. }     25. ToggleAppBarButton(!isPinned, sender as AppBarButton);     26. } |

Code Sample 58. Method for pinning a secondary tile to the Start screen

1. Now we need to add the **Loaded** event for the app bar button that executes when the button has completed loading. Put this event handler in ProductDetailsPage.xaml.cs

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method is called when the PinToStart button has completed loading     3. /// </summary>     4. private void PinToStart\_Loaded(object sender, RoutedEventArgs e)     5. {     6. ToggleAppBarButton(!SecondaryTile.Exists(product.Id), sender as AppBarButton);     7. } |

Code Sample 59. Event handler for Loaded event of Pin To Start button

1. Now it’s time to test it out. Build and run the application and summon the app bar on the product details page by right-clicking on the page (or swiping from the upper or lower edge, as the case may be).

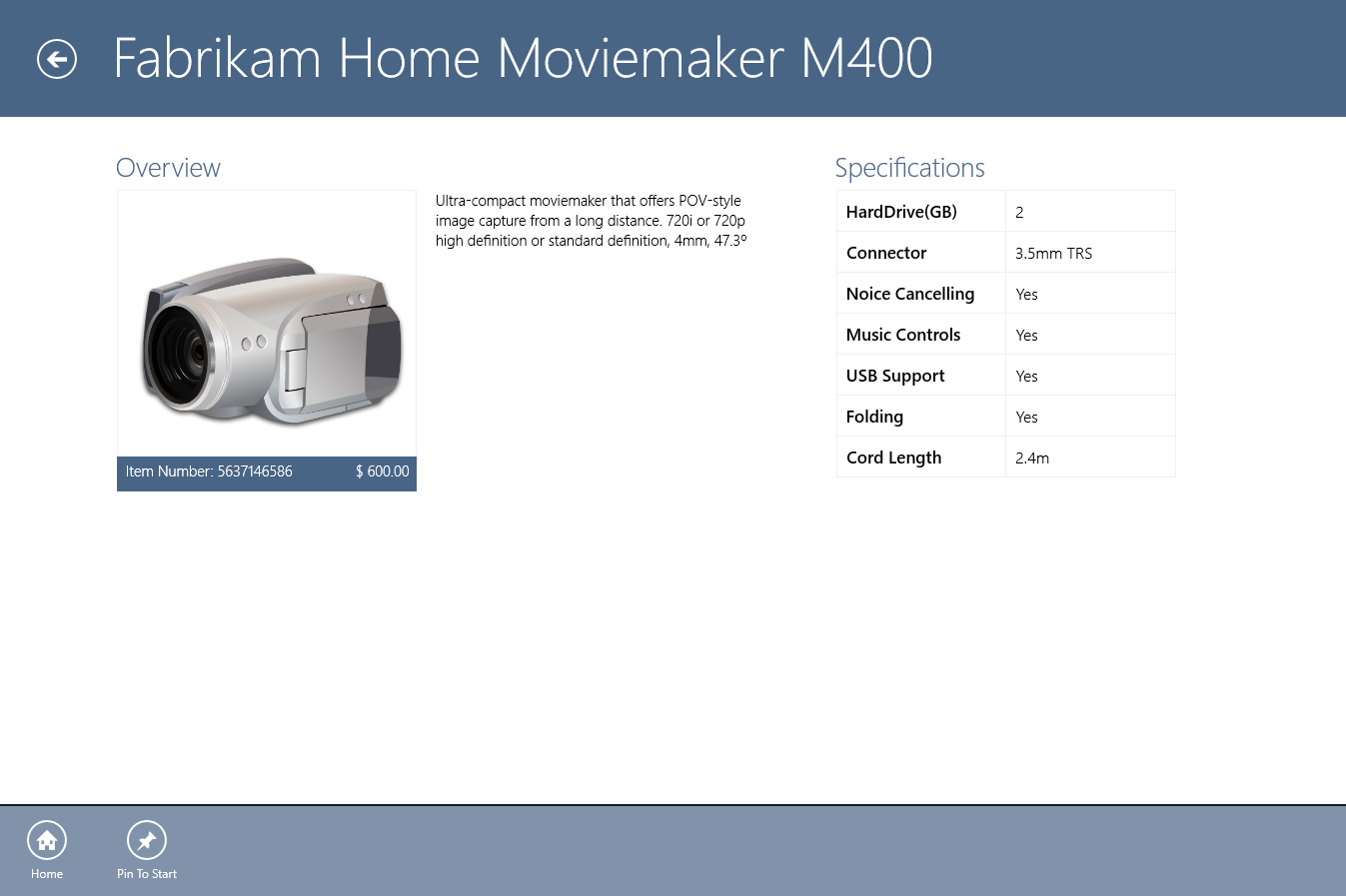


Figure 30. App bar with Pin To Start button enabled

1. Click the **Pin To Start** button, which should correctly display the sample tile.

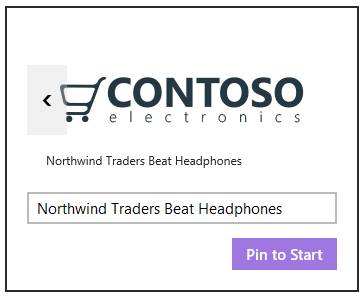


Figure 31. The placeholder for the secondary tile

### Task 4: Enable Secondary Tile Unpinning

Now you need to make it possible to unpin a tile that had been previously pinned to the Start screen.

1. Uncomment the call to the unpin method in the **SecondaryTileCreation** method in the ProductDetailsPage.xaml.cs file, as shown below.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method checks if a secondary tile is already present for this product. If it is     3. /// already present, the product is unpinned, if not the product is pinned.     4. /// </summary>     5. private async Task SecondaryTileCreation(object sender)     6. {     7. string appbarTileId = product.Id;     8. if (!SecondaryTile.Exists(appbarTileId))     9. {     10. await PinSecondaryTile(sender, appbarTileId);     11. }     12. else     13. {     14. await UnpinSecondaryTile(sender, appbarTileId);     15. }     16. } |

Code Sample 60. Call the unpin method

1. Then add the code needed to unpin the secondary tile from the Start screen.

|  |
| --- |
| C# |
| * + 1. /// <summary>     2. /// This method unpins the existing secondary tile.     3. /// The user is shown a message informing whether the tile is unpinned successfully     4. /// </summary>     5. private async Task UnpinSecondaryTile(object sender, string appbarTileId)     6. {     7. SecondaryTile secondaryTile = new SecondaryTile(appbarTileId);     8. bool isUnpinned = await secondaryTile.RequestDeleteForSelectionAsync( GetElementRect((FrameworkElement)sender), Placement.Above);     9. if (isUnpinned)     10. {     11. MessageDialog dialog = new MessageDialog("Product " + product.Name + " successfully unpinned.");     12. await dialog.ShowAsync();     13. }     14. else     15. {     16. MessageDialog dialog = new MessageDialog("Product " + product.Name + " not unpinned.");     17. await dialog.ShowAsync();     18. }     19. ToggleAppBarButton(isUnpinned, sender as AppBarButton);     20. } |

Code Sample 61. Enable unpinning of a product from the Start screen

1. Run the application and select a product that’s already been pinned. Right-click to evoke the app bar. You should see that the style of the button has changed along with the text label.



Figure 32. App bar now showing an Unpin button

1. Click the **Unpin** button.

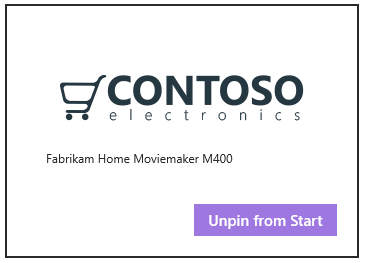


Figure 33. Image of the tile selected for unpinning

### Task 5: Enable Entry Through the Secondary Tile

1. To enable entering the application through the secondary tile, you need to replace code in the **OnLaunched** method of the App.xaml.cs file (Code Sample 62) with custom code (Code Sample 63).

|  |
| --- |
| C# |
| * + 1. if (rootFrame.Content == null)  {      // When the navigation stack isn't restored navigate to the first page,      // configuring the new page by passing required information as a navigation      // parameter      if (!rootFrame.Navigate(typeof(CategoryListingPage), e.Arguments))      {          throw new Exception("Failed to create initial page");      }  } |

Code Sample 62. Code to replace in the OnLaunched method of App.xaml.cs

|  |
| --- |
| C# |
| * + 1. if (rootFrame.Content == null || !string.IsNullOrEmpty(e.Arguments))  {      // When the navigation stack isn't restored navigate to the first page,      // configuring the new page by passing required information as a navigation      // parameter      if (string.IsNullOrEmpty(e.Arguments))      {          if (!rootFrame.Navigate(typeof(CategoryListingPage), e.Arguments))          {              throw new Exception("Failed to create initial page");          }      }      else      {          if (!rootFrame.Navigate(typeof(ProductDetailsPage), e.Arguments))          {              throw new Exception("Failed to create initial page");          }      }  } |

Code Sample 63. Replacement code enabling entry via the secondary tile

1. Finally, check your work. Go to the Start screen and click a secondary tile you created. It will open the product details page for the pinned product.



Figure 34. Secondary tile on the Start screen

# Conclusion

## Lab Completion

### Achievement Summary

Congratulations! You’ve just learned the fundamentals of building a Windows Store app, from how you bind data to the app and customize its look-and-feel with your branding, to the steps involved in implementing several key features, including:

* Search
* Share
* App bar
* Live tiles

#### The flexibility to develop what you need, the way you want

These lab exercises have been kept relatively short and simple on purpose so that you can more easily understand the basic structures and techniques of Windows Store app development before you turn to the task of building the next “killer app” for your business.

But you may also agree after working with it that, even at its most basic, the Windows Store-style app affords a great deal of variety in expression, suited to a wide range of scenarios, while maintaining design and functional consistency to deliver a better overall user experience. And although we chose C# and XAML for this lab, how you prefer to develop within the Windows 8 framework is up to you—whether using JavaScript with HTML/CSS; C#, VB, or C++ with XAML; or C++ with DirectX. The choice is yours.

So if you need to improve field operations, find a better way of presenting your catalogue of products to your customers, help your sales associates be more efficient in managing their clients, or wish to solve some other business problem entirely through the power of software, the skills you’ve acquired today plus the ones you brought with you can provide the foundation for imagining and creating innovative solutions based on Windows 8/8.1 that will serve to differentiate your business—and enhance your career.

Thank you for participating today and good luck!

# Appendix

## Table of Figures

[Figure 1. The structure of the Contoso Electronics solution 6](#_Toc367103465)

[Figure 2. Simple wireframe showing how the hierarchical navigation system works 8](#_Toc367103466)

[Figure 3. Welcome page for the Contoso Electronics app 8](#_Toc367103467)

[Figure 4. Class diagram of the Contoso Electronics data model 13](#_Toc367103468)

[Figure 5. Folders and files related to data binding 17](#_Toc367103469)

[Figure 6. Contents of the Assets folder 20](#_Toc367103470)

[Figure 7. Configure image asset sizes 21](#_Toc367103471)

[Figure 8. Data template for subcategory items 24](#_Toc367103472)

[Figure 9. Data template for product items 25](#_Toc367103473)

[Figure 10. Search charm scope selector 26](#_Toc367103474)

[Figure 11. Search declaration 27](#_Toc367103475)

[Figure 12. Search charm scope selector now displaying the app 27](#_Toc367103476)

[Figure 13. Add search results page 28](#_Toc367103477)

[Figure 14. Search results page (default view) 29](#_Toc367103478)

[Figure 15. Search results page 31](#_Toc367103479)

[Figure 16. Search results page with the proper template 32](#_Toc367103480)

[Figure 17. The completed search results page 36](#_Toc367103481)

[Figure 18. Default Share charm behavior 37](#_Toc367103482)

[Figure 19. Sample email with a screen capture of the current page as an attachment 37](#_Toc367103483)

[Figure 20. Sharing options for product details page 39](#_Toc367103484)

[Figure 21. Intermediate state of share without the content defined 39](#_Toc367103485)

[Figure 22. New options for sharing product details page after content definition 40](#_Toc367103486)

[Figure 23. Sample e-mail with the content defined 41](#_Toc367103487)

[Figure 24. App bar with a navigational command button 43](#_Toc367103488)

[Figure 25. Live tile for the Contoso Electronics app 46](#_Toc367103489)

[Figure 26. Zoomed-out view in the Contoso Electronics app 50](#_Toc367103490)

[Figure 27. Changing the minimum window width in the manifest 51](#_Toc367103491)

[Figure 28. Default view at half screen width 55](#_Toc367103492)

[Figure 29. App in a narrow view with a vertically scrolling list 55](#_Toc367103493)

[Figure 30. App bar with Pin To Start button enabled 59](#_Toc367103494)

[Figure 31. The placeholder for the secondary tile 60](#_Toc367103495)

[Figure 32. App bar now showing an Unpin button 61](#_Toc367103496)

[Figure 33. Image of the tile selected for unpinning 61](#_Toc367103497)

[Figure 34. Secondary tile on the Start screen 62](#_Toc367103498)

## Table of Code Samples

[Code Sample 1. Resource dictionaries and theme dictionaries 12](#_Toc367109677)

[Code Sample 2. The Category.xml schema 14](#_Toc367109678)

[Code Sample 3. The Product.xml schema 15](#_Toc367109679)

[Code Sample 4. Add private method to fetch XML data 18](#_Toc367109680)

[Code Sample 5. Add public method to fetch list of categories 18](#_Toc367109681)

[Code Sample 6. Add public method to fetch category details 19](#_Toc367109682)

[Code Sample 7. Add public method to fetch subcategory name 19](#_Toc367109683)

[Code Sample 8. Bind the product categories 19](#_Toc367109684)

[Code Sample 9. Bind subcategory names 19](#_Toc367109685)

[Code Sample 10. Overriding default SolidColorBrush resources 22](#_Toc367109686)

[Code Sample 11. Adding new SolidColorBrush and ImageBrush resources 22](#_Toc367109687)

[Code Sample 12. Standard framework TextBlock styles 23](#_Toc367109688)

[Code Sample 13. Overriding framework TextBlock styles 24](#_Toc367109689)

[Code Sample 14. Customize styles in the subcategory item data template 24](#_Toc367109690)

[Code Sample 15. Customize styles in the product item data template 25](#_Toc367109691)

[Code Sample 16. Handler method activated on search 29](#_Toc367109692)

[Code Sample 17. Search product data source method 30](#_Toc367109693)

[Code Sample 18. Add private class members 30](#_Toc367109694)

[Code Sample 19. Load state method for search results page 31](#_Toc367109695)

[Code Sample 20. Add required namespace 31](#_Toc367109696)

[Code Sample 21. Assign the correct template 32](#_Toc367109697)

[Code Sample 22. Change the item container height 32](#_Toc367109698)

[Code Sample 23. Filter check event handler 33](#_Toc367109699)

[Code Sample 24. Item click event 33](#_Toc367109700)

[Code Sample 25. Item click event handler 34](#_Toc367109701)

[Code Sample 26. Define the styles for the parent grid 34](#_Toc367109702)

[Code Sample 27. Applying page header styles 35](#_Toc367109703)

[Code Sample 28. Applying styles to the items control 35](#_Toc367109704)

[Code Sample 29. Name the page 35](#_Toc367109705)

[Code Sample 30. Add namespace to ProductDetailsPage.xaml.cs 38](#_Toc367109706)

[Code Sample 31. Add Handler for DataTransferManager.DataRequested event 38](#_Toc367109707)

[Code Sample 32. Remove Handler for DataTransferManager.DataRequested event 38](#_Toc367109708)

[Code Sample 33. Add DataRequested handler 39](#_Toc367109709)

[Code Sample 34. Create content in HTML format for sharing 40](#_Toc367109710)

[Code Sample 35. Adding an app bar in ProductDetailsPage.xaml 42](#_Toc367109711)

[Code Sample 36. Add command button to the app bar 42](#_Toc367109712)

[Code Sample 37. Add event handler for app bar button 43](#_Toc367109713)

[Code sample 38. Generate XML document for values to be displayed in live tile 44](#_Toc367109714)

[Code Sample 39. Add method for fetching random products 45](#_Toc367109715)

[Code Sample 40. Create an XML document for each product 45](#_Toc367109716)

[Code Sample 41. Call the method to create live tiles 45](#_Toc367109717)

[Code Sample 42. Enable tile to queue notifications 46](#_Toc367109718)

[Code Sample 43. Add the Semantic Zoom control 48](#_Toc367109719)

[Code Sample 44. Move GridView inside ZoomedInView 48](#_Toc367109720)

[Code Sample 45. Item template for zoomed-out GridView items 49](#_Toc367109721)

[Code Sample 46. Add zoomed-out view 49](#_Toc367109722)

[Code Sample 47. Assign data source to the zoomed-out view 49](#_Toc367109723)

[Code Sample 48. ListView control configured for a narrow, "snapped" view 52](#_Toc367109724)

[Code Sample 49. Setting up visual states for the page 53](#_Toc367109725)

[Code Sample 50. Register a handler for the Window.SizeChanged event 53](#_Toc367109726)

[Code Sample 51. Method for changing visual states 54](#_Toc367109727)

[Code sample 52. LoadState method with call to UpdateVisualState 54](#_Toc367109728)

[Code Sample 53. Adding PinToStart in ProductDetailsPage.xaml 56](#_Toc367109729)

[Code Sample 54. Adding the handler for the click event 57](#_Toc367109730)

[Code Sample 55. Add method to check pin state for given product 57](#_Toc367109731)

[Code Sample 56. Create secondary tile placeholder 58](#_Toc367109732)

[Code Sample 57. Enable toggling of the Pin To Start button 58](#_Toc367109733)

[Code Sample 58. Method for pinning a secondary tile to the Start screen 59](#_Toc367109734)

[Code Sample 59. Event handler for Loaded event of Pin To Start button 59](#_Toc367109735)

[Code Sample 60. Call the unpin method 60](#_Toc367109736)

[Code Sample 61. Enable unpinning of a product from the Start screen 61](#_Toc367109737)

[Code Sample 62. Code to replace in the OnLaunched method of App.xaml.cs 62](#_Toc367109738)

[Code Sample 63. Replacement code enabling entry via the secondary tile 62](#_Toc367109739)

1. Hereafter “App Clinic,” for short. [↑](#footnote-ref-2)