

Hand­s-on lab

Lab 5: Integrating Ads

May 2015

Contents

[Overview 3](#_Toc420065245)

[Exercise 1: Introducing Ads to the App 4](#_Toc420065246)

[Task 1 – Open the starter project 4](#_Toc420065247)

[Task 2 – Install the Windows 10 Advertising SDK 5](#_Toc420065248)

[Task 3 – Add an interstitial ad 7](#_Toc420065249)

[Task 4 – Show an inline ad 15](#_Toc420065250)

[Exercise 2: Integrate In-App Product Purchases 17](#_Toc420065251)

[Task 1 – Open the starter project 17](#_Toc420065252)

[Task 2 – Add the InAppPurchaseService 18](#_Toc420065253)

[Task 3 – Update the view model 19](#_Toc420065254)

[Task 4 – Update the view 21](#_Toc420065255)

[Summary 24](#_Toc420065256)

Overview

In this lab, you will install the Windows 10 Advertising SDK, implement interstitial and inline ads, and integrate in-app product purchases. This will also be the first time you see the Contoso Stack sample application. In this lab we dive straight into adding features direct to the app – in furture labs we will review the existing implementation in more detail.

# Objectives

* 1. This lab will show you how to:
  + Install the Windows 10 Advertising SDK
  + Introduce interstitial and inline ads to the app
  + Integrate in-app product purchases and use them to remove ads from the app

# System requirements

* 1. You must have the following to complete this lab:
  + Microsoft Windows 10
  + Microsoft Visual Studio 2015

# Setup

* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install Microsoft Windows 10.
  3. Install Microsoft Visual Studio 2015.
  4. Install the Universal Windows Platform SDK.

# Exercises

* 1. This Hands-on lab includes the following exercises:
  2. Introducing Ads to the App
  3. Integrate In-App Product Purchases
  4. Estimated time to complete this lab:  **45 to 60 minutes**.

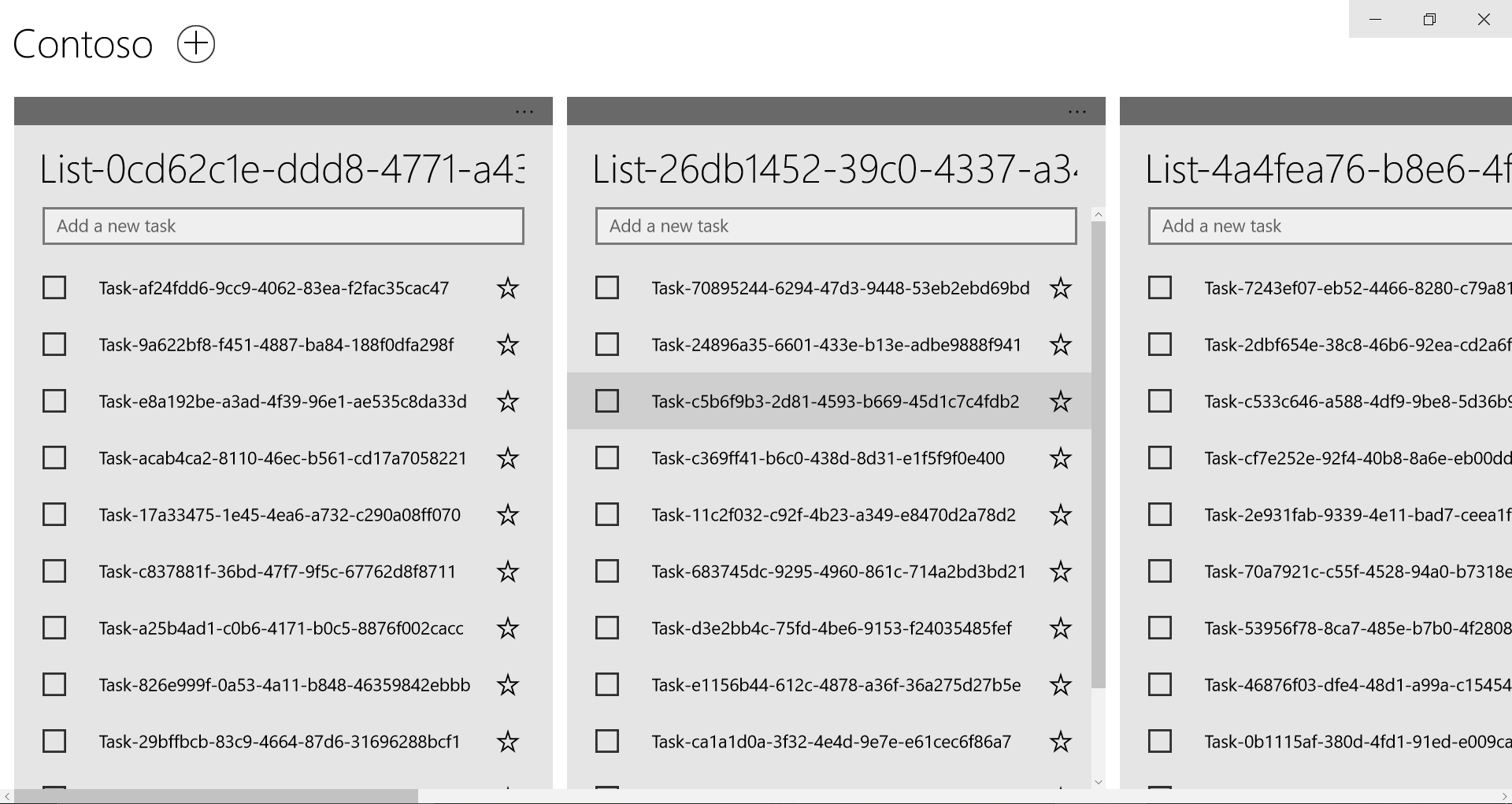
Exercise 1: Introducing Ads to the App

1. In this exercise, you will install the Windows 10 Advertising SDK and use it to add interstitial and inline ads to your app.

Task 1 – Open the starter project

* 1. The first task is to open the starter project we have created for you.

1. Open the file location where you installed the hands-on labs. Navigate to the **\Lab 5\Solution\Advertising** folder and open **ContosoStack.sln** in Visual Studio 2015.
2. To prepare to build and run your app, use the Solution Configurations dropdown to choose the Debug configuration and use the Solution Platforms dropdown to target x86 (Figure 8). To run the app on the local machine, select Local Machine from the drop-down list next to the Start Debugging button on the debugger Standard toolbar.
3. Build and run the app.

**

**Figure 1**

*The Advertising starter app.*

Return to Visual Studio and stop debugging.

Task 2 – Install the Windows 10 Advertising SDK

Before adding ads to your app, you will first need to install the Windows 10 Advertising SDK.

1. Begin by closing Visual Studio 2015 if it is open. All instances of Visual Studio must be closed for the Advertising SDK to install properly.
2. In your browser, navigate to <https://visualstudiogallery.msdn.microsoft.com/401703a0-263e-4949-8f0f-738305d6ef4b> . Download the Windows 10 (Universal Windows Platform) preview by selecting **Download**.

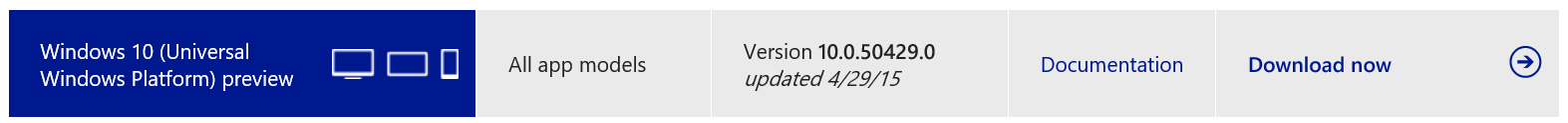


Figure 2

Download the Windows 10 Advertising SDK.

1. Run the msi file when prompted by the browser. When the Microsoft Advertising setup launches, accept the terms of the license agreement and choose **Install**. If prompted by the User Account Control, select **Yes** to allow the app to install software on your PC. When the installation is complete, use the **Finish** button to exit the setup wizard.

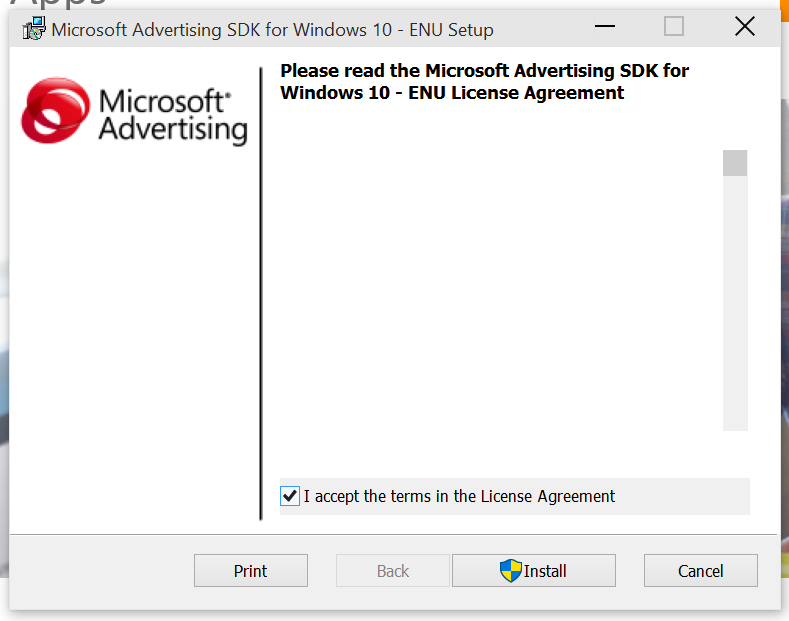


Figure 3

Setup wizard for the Windows 10 Advertising SDK.

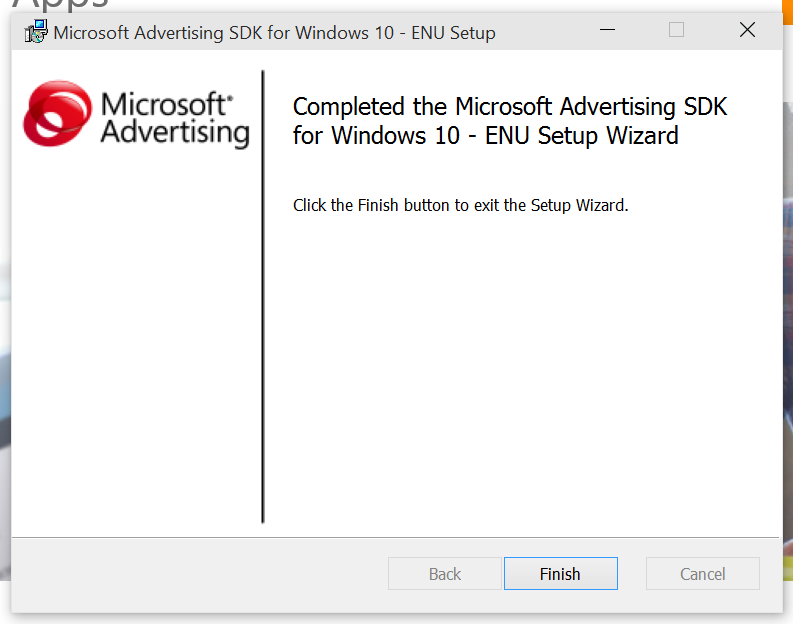


Figure 4

Finish installation of the Windows 10 Advertising SDK.

1. Reopen Visual Studio and load the Advertising project you created in Exercise 1. When the project is open, right-click References in the Solutions Explorer and choose **Add Reference**.

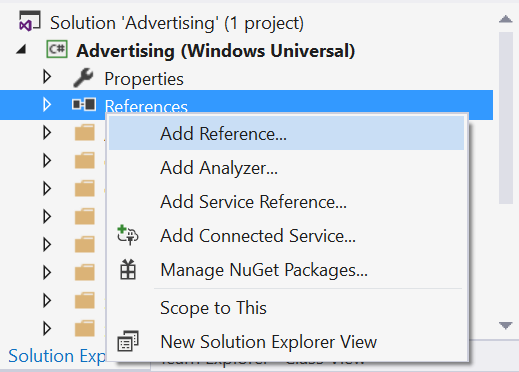


Figure 5

Navigate to the Add Reference dialog.

1. Expand the Windows Universal section and select **Extensions**. You will see a list of SDKs applicable to your project. Check the box next to the **Microsoft Advertising SDK for XAML** to select it and click OK to add it to the project as a reference.

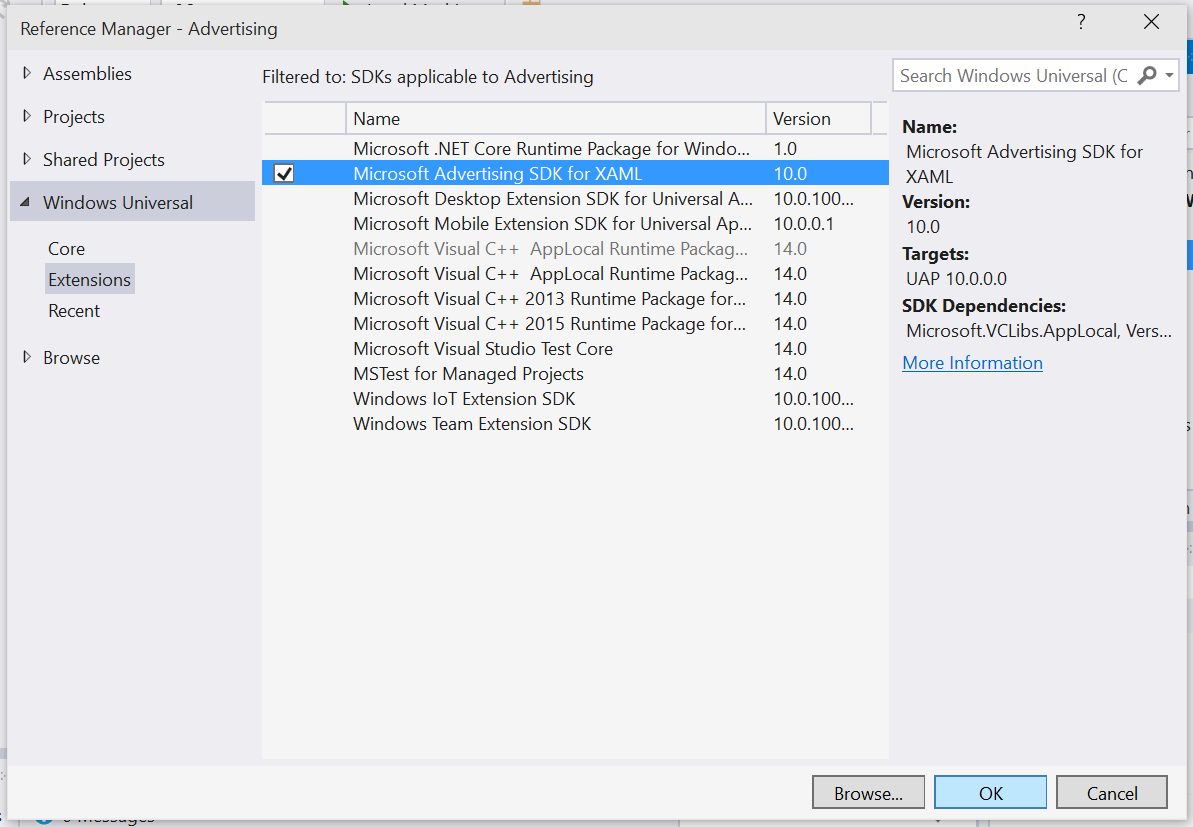


Figure 6

Add the Microsoft Advertising SDK for XAML as a project reference.

1. When the Add Reference dialog closes, you will see the Microsoft Advertising SDK appear in the list of project references.

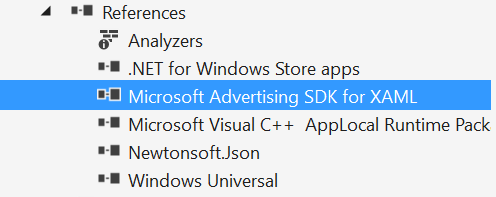


Figure 7

Add the Microsoft Advertising SDK for XAML as a project reference.

Task 3 – Add an interstitial ad

* 1. Now that you have referenced the Microsoft Advertising SDK in the Advertising project, you can begin to integrate ads into your app. In this task, you will create a new class called DemoAds, which will use test AppIds and AdUnits provided by Microsoft to display an interstitial ad in your app.

1. To create the new DemoAds class, right-click on the **Models** folder in the Solutions Explorer and choose **Add > New Item**. When the Add New Item dialog appears, select Visual C# Class as your new item type (Figure 17). Give the class the name **DemoAds.cs** and use the Add button to create it.

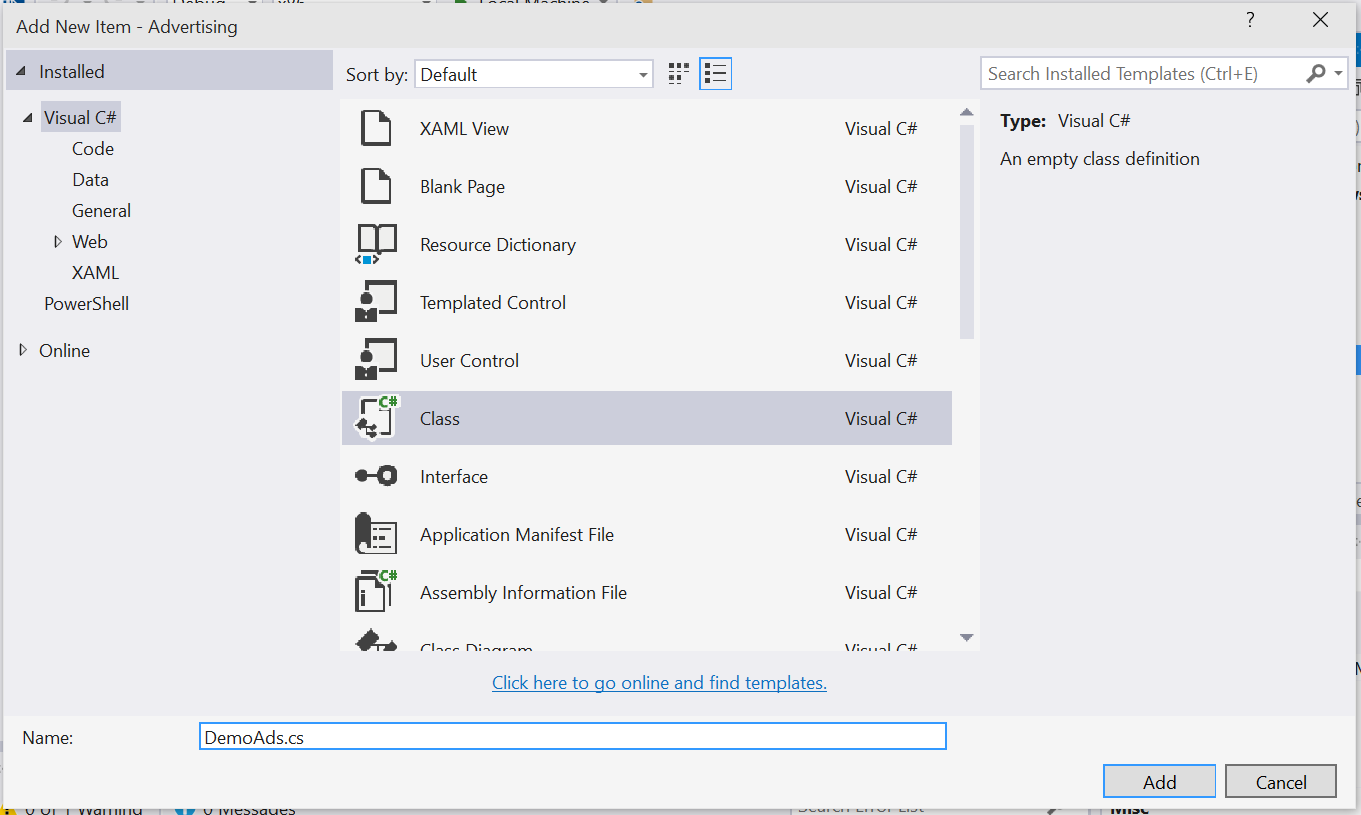


Figure 8

Create a new Visual C# class called DemoAds.

1. Open DemoAds.cs to view the class definition. In this step, you will replace the empty class definition with working DemoAds and AdUnit classes. The code block below shows the starting point that Visual Studio provides for you.

C#

* 1. using System;
  2. using System.Collections.Generic;
  3. using System.Linq;
  4. using System.Text;
  5. using System.Threading.Tasks;
  6. namespace Advertising.Models
  7. {

   class DemoAds

   {

   }

* 1. }

Replace the empty class definition in the previous code block with the following class definition shown in red. Save DemoAds.cs.

C#

* 1. using System;
  2. using System.Collections.Generic;
  3. using System.Linq;
  4. using System.Text;
  5. using System.Threading.Tasks;
  6. namespace Advertising.Models
  7. {

   /\*

       These demo ad values are drawn from: <https://msdn.microsoft.com/en-US/library/mt125365(v=msads.100).aspx>

   \*/

   public static class DemoAds

   {

       public static Dictionary<string, AdUnit> ImageAdUnits { get; private set; }

       public static AdUnit VideoAdUnit { get; private set; }

       static DemoAds()

       {

           ImageAdUnits = new Dictionary<string, AdUnit>();

           ImageAdUnits.Add("300 x 50",

               new AdUnit { Size = "300 x 50", AdUnitId = "10865275", AppId = "3f83fe91-d6be-434d-a0ae-7351c5a997f1" });

           ImageAdUnits.Add("320 x 50",

               new AdUnit { Size = "320 x 50", AdUnitId = "10865270", AppId = "3f83fe91-d6be-434d-a0ae-7351c5a997f1" });

           ImageAdUnits.Add("300 x 250",

               new AdUnit { Size = "300 x 250", AdUnitId = "10043121", AppId = "d25517cb-12d4-4699-8bdc-52040c712cab" });

           ImageAdUnits.Add("300 x 600",

               new AdUnit { Size = "300 x 600", AdUnitId = "10043122", AppId = "d25517cb-12d4-4699-8bdc-52040c712cab" });

           ImageAdUnits.Add("480 x 80",

               new AdUnit { Size = "480 x 80", AdUnitId = "10865272", AppId = "3f83fe91-d6be-434d-a0ae-7351c5a997f1" });

           ImageAdUnits.Add("640 x 100",

               new AdUnit { Size = "640 x 100", AdUnitId = "10865273", AppId = "3f83fe91-d6be-434d-a0ae-7351c5a997f1" });

           ImageAdUnits.Add("728 x 90",

               new AdUnit { Size = "728 x 90", AdUnitId = "10043123", AppId = "d25517cb-12d4-4699-8bdc-52040c712cab" });

           VideoAdUnit = new AdUnit { Size = "Video", AdUnitId = "11389925", AppId = "d25517cb-12d4-4699-8bdc-52040c712cab" };

       }

   }

   public class AdUnit

   {

       public string Size { get; set; }

       public string AdUnitId { get; set; }

       public string AppId { get; set; }

   }

* 1. }

1. Open **ViewModels > MainPageViewModel.cs**. Add the following variables shown in red to your viewmodel and save the file:

C#

* 1. bool \_busy = false;
  2. public bool Busy { get { return \_busy; } set { Set(ref \_busy, value); }
  3. }

bool \_showAds = true;

public bool ShowAds { get { return \_showAds; } set { Set(ref \_showAds, value); } }

bool \_viewedFullInterstitial = true;

public bool ViewedFullInterstitial { get { return \_viewedFullInterstitial; } set { Set(ref \_viewedFullInterstitial, value); } }

private ObservableCollection<ViewModels.TodoListViewModel> \_TodoLists = new ObservableCollection<TodoListViewModel>();

1. Now that you have created the DemoAds model, you can reference it in the view. In the Solutions Explorer, navigate to the **Views** directory and open **MainPage.xaml.cs**. Add the following namespaces shown in red to reference DemoAds and the Microsoft Advertising UI.

C#

* 1. using System;
  2. using Windows.ApplicationModel.Core;
  3. using Windows.UI.Xaml;
  4. using Windows.UI.Xaml.Controls;
  5. using Microsoft.Advertising.WinRT.UI;
  6. using ContosoStack.Models;
  7. namespace ContosoStack.Views
  8. {

1. After the class definition, insert the private InterstitialAd variable.

C#

* 1. namespace AdsInTemplate10.Views
  2. {
  3. public sealed partial class MainPage : Page
  4. {
  5. private InterstitialAd \_interstitialAd;
  6. public MainPage()

1. Initialize the interstitial class and wire up the AdReady, Cancelled, Completed, and ErrorOccurred events in the constructor.

C#

* 1. public MainPage()
  2. {
  3. this.InitializeComponent();
  4. this.ViewModel = this.DataContext as ViewModels.MainPageViewModel;
  5. CoreApplication.GetCurrentView().TitleBar.ExtendViewIntoTitleBar = true;
  6. Window.Current.SetTitleBar(AppTitle);
  7. if (this.ViewModel.ShowAds)
  8. {
  9. // initialize the interstitial class
  10. \_interstitialAd = new InterstitialAd();
  11. // wire up all 4 events
  12. \_interstitialAd.AdReady += interstitialAd\_AdReady;
  13. \_interstitialAd.Cancelled += interstitialAd\_Cancelled;
  14. \_interstitialAd.Completed += interstitialAd\_Completed;
  15. \_interstitialAd.ErrorOccurred += interstitialAd\_ErrorOccurred;
  16. RequestAd();
  17. }
  18. else
  19. {
  20. // start normally
  21. }
  22. }

1. Create the AdReady, Cancelled, Completed, and ErrorOccurred events beneath the constructor definition.

C#

* 1. RequestAd();
  2. }
  3. else
  4. {
  5. // start normally
  6. }
  7. }
  8. private void interstitialAd\_ErrorOccurred(object sender, AdErrorEventArgs e)
  9. {
  10. // handle errors here
  11. }
  12. private void interstitialAd\_Completed(object sender, object e)
  13. {
  14. // raised when the user has watched the full video
  15. }
  16. private void interstitialAd\_Cancelled(object sender, object e)
  17. {
  18. // raised if the user interrupts the video
  19. }
  20. private void interstitialAd\_AdReady(object sender, object e)
  21. {
  22. // raised when an ad is ready to show
  23. }

ViewModels.MainPageViewModel ViewModel { get; set; }

1. Add the definition for the RequestAd event under the constructor.

C#

* 1. RequestAd();
  2. }
  3. else
  4. {
  5. // start normally
  6. }
  7. }
  8. private void RequestAd()
  9. {
  10. \_interstitialAd.RequestAd(AdType.Video, DemoAds.VideoAdUnit.AppId, DemoAds.VideoAdUnit.AdUnitId);
  11. }
  12. private void interstitialAd\_ErrorOccurred(object sender, AdErrorEventArgs e)
  13. {
  14. Note: For the purposes of this lab, we will display the interstitial as soon as the ad is ready by adding it to the AdReady event handler.

1. Add the definition for the ErrorOccurred event. Be sure to add **async** to the event handler to accommodate the await.
   * 1. C#

private async void interstitialAd\_ErrorOccurred(object sender, AdErrorEventArgs e)

* 1. {
  2. // handle errors here
  3. var dialog = new ContentDialog
  4. {
  5. Title = "An Error",
  6. Content = e.ErrorMessage,
  7. PrimaryButtonText = "OK",
  8. IsPrimaryButtonEnabled = true
  9. };
  10. await dialog.ShowAsync();
  11. }
  12. **Note:** An enterprise-level application will require more robust error handling than we provide here.

1. Add the definition for the AdReady event.

C#

* 1. private void interstitialAd\_AdReady(object sender, object e)
  2. {
  3. //raised when an ad is ready to show
  4. // This is just for demoing - you should handle this differently in a production app
  5. if (\_interstitialAd.State == InterstitialAdState.Ready)
  6. {
  7. \_interstitialAd.Show();
  8. }
  9. }

1. Build and run the app. You will see an interstitial video ad play when the app loads.
2. We will now add the capability to ensure the user watches an entire video ad before proceeding. We will leverage the Cancelled event by adding the following:

C#

private void interstitialAd\_Completed(object sender, object e)

* 1. {
  2. // raised when the user has watched the full video
  3. this.ViewModel.ViewedFullInterstitial = true;
  4. }
  5. private async void interstitialAd\_Cancelled(object sender, object e)
  6. {
  7. // raised if the user interrupts the video
  8. var dialog = new ContentDialog
  9. {
  10. Title = "Ad Interrupted",
  11. Content = "You must watch the complete ad!",
  12. PrimaryButtonText = "OK",
  13. IsPrimaryButtonEnabled = true
  14. };
  15. await dialog.ShowAsync();
  16. RequestAd();
  17. }
  18. **Note:** As we are going to use a ContentDialog control to display the error message via an async method, we must be sure to add async to the method definition of the interstitialAd\_Cancelled event handler. The ContentDialog is a new control in Windows 10 that makes it easier to display rich content via an app modal dialog.

1. Build and run the app. Clicking the running video will display a back button. Select the back button while viewing the ad to see the behavior of the interstitialAd\_Cancelled event.
2. Moving forward, it would be inconvenient to be unable to skip the ad during these exercises. To give yourself the option to skip the ad, comment out the contents of the cancelled event handler.

C#

private async void interstitialAd\_Cancelled(object sender, object e)

* 1. {
  2. // raised if the user interrupts the video
  3. //var dialog = new ContentDialog
  4. //{
  5. //    Title = "Ad Interrupted",
  6. //    Content = "You must watch the complete ad!",
  7. //    PrimaryButtonText = "OK",
  8. //    IsPrimaryButtonEnabled = true
  9. //};
  10. //await dialog.ShowAsync();
  11. //RequestAd();
  12. }

Build and run the app again. Confirm that you can skip the ad by selecting the back button while it is playing.

1. Task 4 – Show an inline ad
   1. In this task, you will use the Microsoft Advertising SDK to display an inline ad in a separate hub section in your app. The same techniques can be used to place ads inline with content within hub sections or other list controls.
2. Open **Views > MainPage.xaml**. Add the following namespaces shown in red.

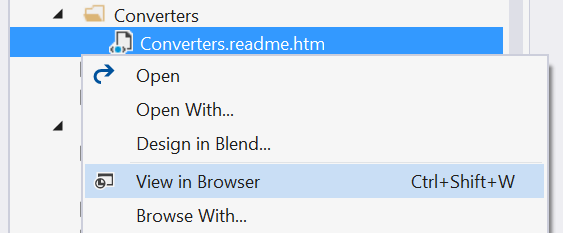
XAML

* 1. xmlns:models="using:ContosoStack.Models"
  2. xmlns:UI="using:Microsoft.Advertising.WinRT.UI"
  3. xmlns:c="using:Advertising.Converters"
  4. x:Name="ThisPage"

1. In order to toggle visibility of the advertising section, we will use the ValueWhenConverter. Add the following code to the XAML ResourceDictionary:

XAML

* 1. <ResourceDictionary>
  2. <ResourceDictionary.MergedDictionaries>
  3. <ResourceDictionary Source="ms-appx:///Styles/StarRadioButtonStyle.xaml"/>
  4. </ResourceDictionary.MergedDictionaries>
  5. <c:ValueWhenConverter x:Key="VisibleWhenTrueConverter">
  6. <c:ValueWhenConverter.When>
  7. <x:Boolean>True</x:Boolean>
  8. </c:ValueWhenConverter.When>
  9. <c:ValueWhenConverter.Value>
  10. <Visibility>Visible</Visibility>
  11. </c:ValueWhenConverter.Value>
  12. <c:ValueWhenConverter.Otherwise>
  13. <Visibility>Collapsed</Visibility>
  14. </c:ValueWhenConverter.Otherwise>
  15. </c:ValueWhenConverter>
  16. </ResourceDictionary>
  17. **Note:** To learn more about this converter, navigate to the **Converters** directory in the Solutions Explorer. Right-click **Converters.readme.htm** and select View in Browser to read the documentation.

1. 
2. Figure 9
3. View the Template 10 converters documentation.
4. To create a container for your inline ad, add the following Advertising hub section above the existing hub section in **MainPage.xaml**. This hub section will leverage the **AdControl** from the Advertising SDK to show a static image ad and will display to the left of the app content. In design view, this hub section will display a blue rectangle with the same dimensions as the ad.

XAML

* 1. <HubSection VerticalContentAlignment="Stretch" x:Name="AdvertisingSection"
  2. Header="Advertising"
  3. Visibility="{Binding Path=ShowAds, Converter={StaticResource ResourceKey=VisibleWhenTrueConverter}}">
  4. <DataTemplate>
  5. <Grid VerticalAlignment="Top">
  6. <!-- The rectangle acts as a place holder so we can see where the ad control is located-->
  7. <Rectangle Fill="Blue" Width="300" Height="600"/>
  8. <UI:AdControl ApplicationId="d25517cb-12d4-4699-8bdc-52040c712cab"
  9. AdUnitId="10043122"
  10. Height="600"
  11. VerticalAlignment="Top"
  12. Width="300"/>
  13. </Grid>
  14. </DataTemplate>
  15. </HubSection>

1. Build and run your app. You will see the inline image ad appear to the left of your app content.

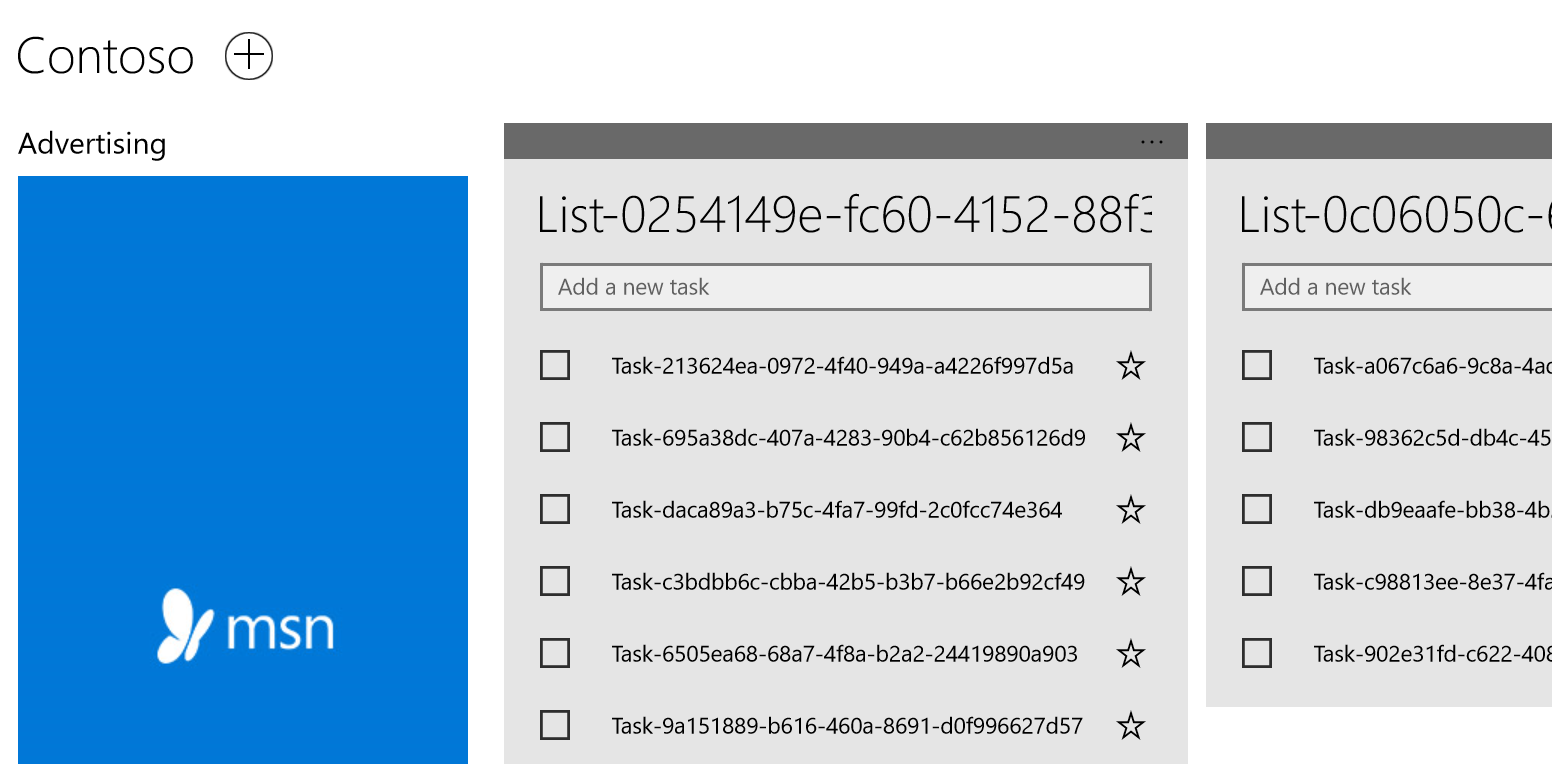


Figure 10

Inline ad in the Advertising app.

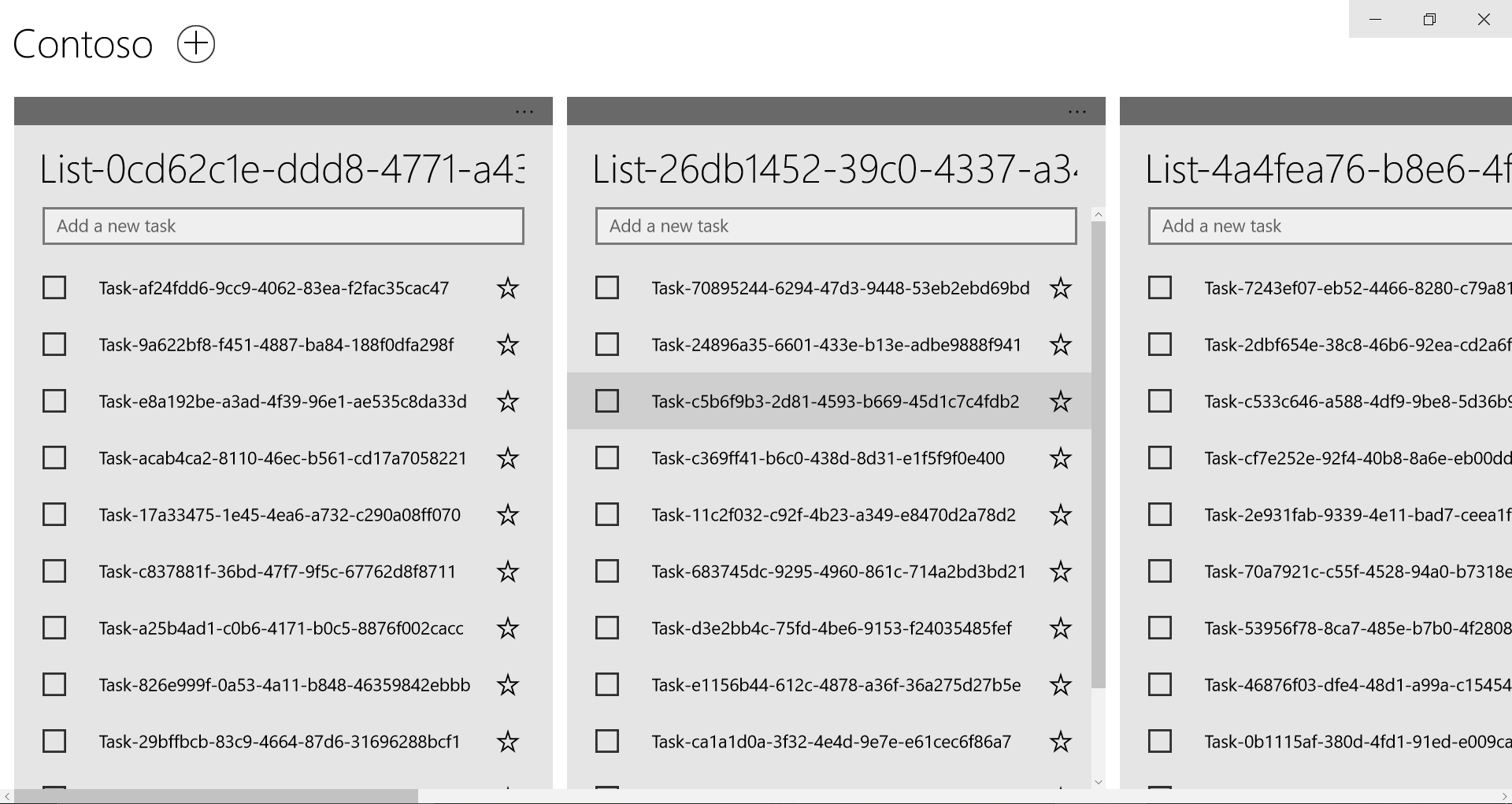
Exercise 2: Integrate In-App Product Purchases

In this exercise, you will give your app users the option to remove ads via an in-app product purchase. A simulator running on your local machine will play the role of the Windows Store.

Task 1 – Open the starter project

* 1. The first task is to open the starter project we have created for you.

1. Open the file location where you installed the hands-on labs. Navigate to the **\Lab 5\Solution\InAppPurchase** folder and open **ContosoStack.sln** in Visual Studio 2015.
2. To prepare to build and run your app, use the Solution Configurations dropdown to choose the Debug configuration and use the Solution Platforms dropdown to target x86 (Figure 8). To run the app on the local machine, select Local Machine from the drop-down list next to the Start Debugging button on the debugger Standard toolbar.
3. Build and run the app.

**

**Figure 11**

*The InAppPurchase starter app.*

Return to Visual Studio and stop debugging.

Task 2 – Add the InAppPurchaseService

* 1. In this task, you will add a service to your project to simulate the Windows Store.
  2. In your Advertising project, right-click on the **Services** directory and create a new folder. Name the folder **InAppPurchaseService**.

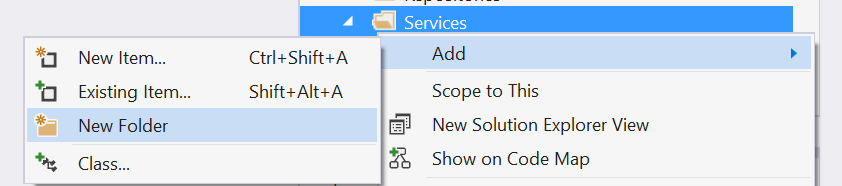


Figure 12

Add a new folder inside the Services directory.

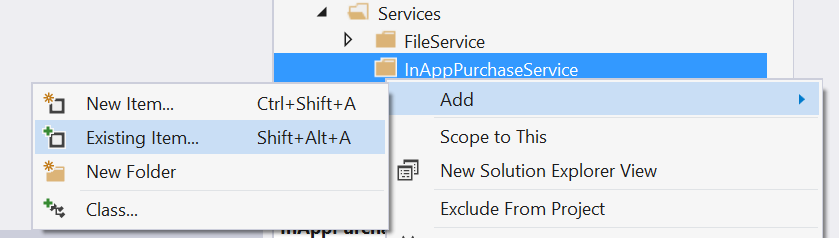
* 1. Right-click on the InAppPurchaseService folder you just created and choose **Add > Existing Item**. Navigate to the \**Assets\Lab 5** folder and add the file **InAppPurchaseService.cs**. The file will appear in the Solution Explorer.
     1. 

Figure 13

Add an existing item to the InAppPurchaseService folder.

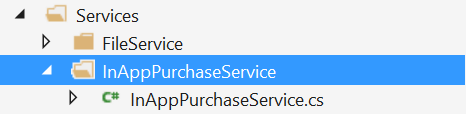
* + 1. 

Figure 14

InAppPurchaseService.cs in the Solution Explorer.

* 1. Open InAppPurchaseService.cs in VisualStudio. Check to make sure the namespace matches your project name. In this lab, we named our project ContosoStack, so the respective namespace is **ContosoStack.Services.InAppPurchaseService**.

C#

* 1. namespace Advertising.Services.InAppPurchaseService
  2. {

If you gave your project a different name, update the namespace accordingly and save the file.

1. Task 3 – Update the view model

Now that the InAppPurchaseService has been added, you will integrate it into your view model.

1. Open **ViewModels > MainPageViewModel.cs**. Begin by adding the Services namespace for your project.
   * 1. C#
2. using Windows.UI.Xaml.Navigation;
   1. using ContosoStack.Services;
3. Create the following member variable.
   * 1. C#
   1. public class MainPageViewModel : Mvvm.ViewModelBase
   2. {
   3. Repositories.TodoListRepository \_todoListRepository;
   4. Services.InAppPurchaseService.InAppPurchaseService \_inAppPurchaseService;
   5. Inside the constructor, instantiate the InAppPurchaseService to purchase the product “NoAds”.
      1. C#
   6. public MainPageViewModel()
   7. {
   8. \_todoListRepository = new Repositories.TodoListRepository();
   9. \_inAppPurchaseService = new Services.InAppPurchaseService.InAppPurchaseService("NoAds");
4. Add the SetupIapCommand beneath the SaveCommand definition.
   * 1. C#
   1. Mvvm.Command \_SetupIapCommand = default(Mvvm.Command);
   2. public Mvvm.Command SetupIapCommand { get { return \_SetupIapCommand ?? (\_SetupIapCommand = new Mvvm.Command(ExecuteSetupIapCommand, CanExecuteSetupIapCommand)); } }
   3. private bool CanExecuteSetupIapCommand() { return true; }
   4. private async void ExecuteSetupIapCommand()
   5. {
   6. while (Busy)
   7. {
   8. await Task.Delay(100);
   9. }
   10. try
   11. {
   12. Busy = true;
   13. await \_inAppPurchaseService.Setup();
   14. // We don't show ads if we have purchased the IAP
   15. ShowAds = !\_inAppPurchaseService.IsPurchased;
   16. }
   17. finally { Busy = false; }
   18. }
5. Add the RemoveAds command below the SetupIapCommand.
   * 1. C#
6. Mvvm.Command \_RemoveAdsCommand = default(Mvvm.Command);
7. public Mvvm.Command RemoveAdsCommand { get { return \_RemoveAdsCommand ?? (\_RemoveAdsCommand = new Mvvm.Command(ExecuteRemoveAdsCommand, CanExecuteRemoveAdsCommand)); } }
8. private bool CanExecuteRemoveAdsCommand() { return ShowAds; }
9. private async void ExecuteRemoveAdsCommand()
10. {
11. try
12. {
13. Busy = true;
14. if (await \_inAppPurchaseService.Purchase())
15. {
16. this.ShowAds = false;
17. }
18. }
19. finally { Busy = false; }
20. }
21. Add the code to refresh the CanExecute state of the RemoveAdsCommand in the MainPageViewModel constructor.
    * 1. C#
    1. // update commands
    2. this.PropertyChanged += (s, e) =>
    3. {
    4. this.AddListCommand.RaiseCanExecuteChanged();
    5. this.RemoveListCommand.RaiseCanExecuteChanged();
    6. this.RemoveAdsCommand.RaiseCanExecuteChanged();
    7. };
22. Add the command to setup the IapPurchaseService on navigation to the MainPageViewModel.
    * 1. C#
    1. public override void OnNavigatedTo(string parameter, NavigationMode mode, Dictionary<string, object> state)
    2. {
    3. LoadCommand.Execute(null);
    4. SetupIapCommand.Execute(null);
    5. }
23. Build your project to ensure there are no errors. If you run your project, you will not see any changes yet. You will update the view in the next task.
24. Task 4 – Update the view

Now that your in-app purchase service and view model are set up, the last task is to add a command to your view so that users can make the product purchase.

1. Open **Views > MainPage.xaml.** Add a bottom app bar beneath the data context to display your purchase button, which you will bind to the RemoveAdsCommand.
   * 1. XAML
   1. <Page.DataContext>
   2. <vm:MainPageViewModel />
   3. </Page.DataContext>
   4. <Page.BottomAppBar>
   5. <CommandBar>
   6. <AppBarButton Icon="Shop" Label="Remove Ads" Command="{Binding RemoveAdsCommand}" />
   7. </CommandBar>
   8. </Page.BottomAppBar>
2. Build and run your app. When you click on the app bar button for the Remove Ads command, a dialog will appear to confirm your simulated purchase. After confirming the purchase, you will see the inline ad disappear. Completing the purchase will also disable the app bar button.

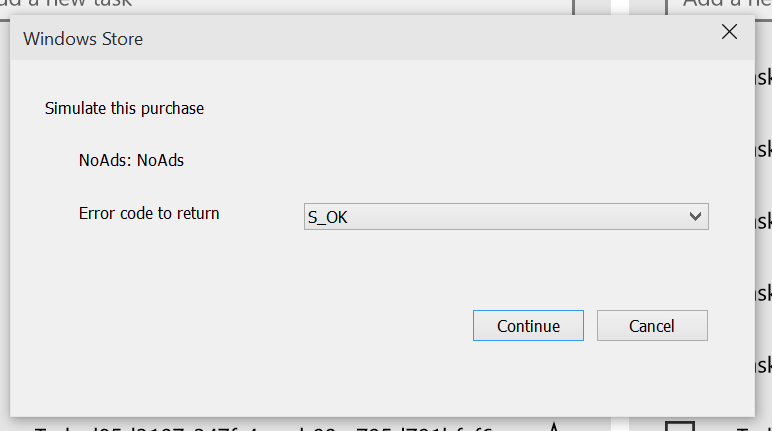


Figure 15

The simulated purchase dialog.

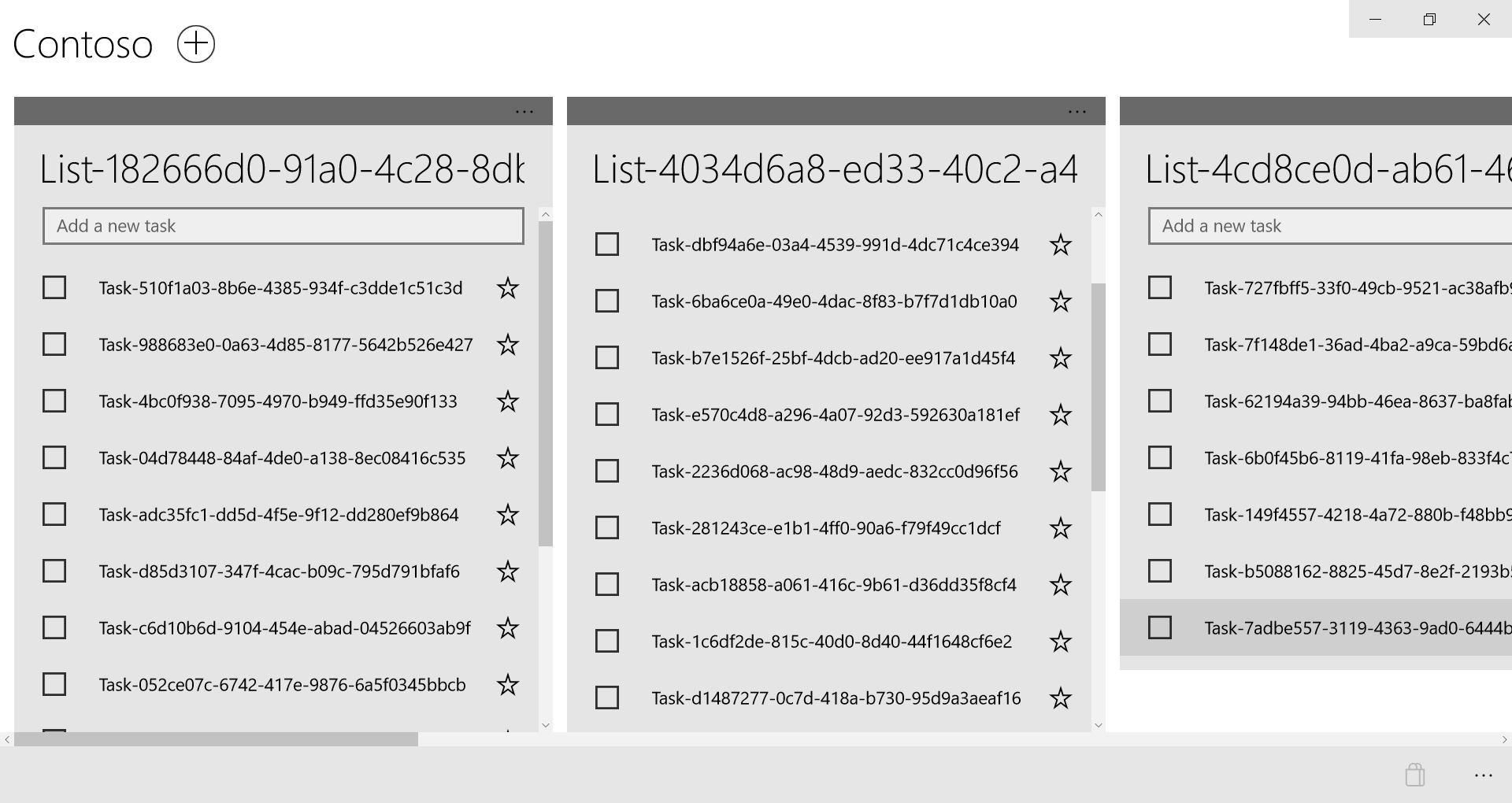


Figure 16

Post-purchase view with the disabled app bar button.

* 1. **Note:** We are not saving state, so the ads will appear again when you rerun your app.

Summary

* 1. In this lab, you downloaded and installed the Advertising SDK and displayed the new Video Interstital Ads. You then added an image based ad into the Hub control and incorporated an In-App Product purchase to disable and hide the ads.
  2. It’s a great start, but there’s still more to do to make a first-class app for the Windows Store. The journey continues in Lab 6!