

Hand­s-on lab

Lab: Integrating Ads

September 2015

Contents

[Overview 3](#_Toc430922459)

[Exercise 1: Introducing Ads to the App 4](#_Toc430922460)

[Task 1 – Create a blank Universal Windows app 4](#_Toc430922461)

[Task 2 – Install the Windows 10 Advertising SDK 6](#_Toc430922462)

[Task 3 – Add an interstitial ad 9](#_Toc430922463)

[Task 4 – Require the ad 15](#_Toc430922464)

[Task 4 – Show an inline ad 16](#_Toc430922465)

[Summary 20](#_Toc430922466)

Overview

1. The Universal Ad SDK is easy to integrate and gives you the ability to promote and monetize your app in markets around the world.

In this lab, you will install the Windows 10 Ad Mediator package, which includes the Microsoft Advertising SDK for XAML. The Microsoft Advertising libraries for XAML/JavaScript are different extensions from the AdMediator control (named Microsoft Advertising Universal SDK version 1.0 in the Visual Studio Reference Manager). For more information, visit <https://msdn.microsoft.com/en-us/library/mt313199(v=msads.30).aspx>.

You will use the Advertising SDK to implement interstitial and inline ads, leveraging the demo ads made available by Microsoft.

# Objectives

* 1. This lab will show you how to:
  + Install the Windows 10 Ad Mediator
  + Introduce an interstitial ad to the app
  + Add an inline ad to the app content
  + Implement a ValueWhenConverter to toggle visibility of the ad

# 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 Windows Ad Mediator Installation package.

*Instructions for installing the Windows Ad Mediator are in Exercise 1: Task 2.*

# Exercises

* 1. This Hands-on lab includes the following exercises:
  2. Introducing Ads to the App
  3. Estimated time to complete this lab:  **30 to 45 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 – Create a blank Universal Windows app

We will begin by creating a project from the Blank App template.

1. In a new instance of Visual Studio 2015, choose **File > New> Project** to open the New Project dialog. Navigate to **Installed > Templates > Visual C# > Windows > Universal** and select the **Blank App (Universal Windows)** template.
2. Name your project **Advertising** and select the file system location where you will save your Hands-on Lab solutions. We have created a folder in our **C:** directory called **HOL** that you will see referenced in screenshots throughout the labs.

Leave the options selected to **Create new solution** and **Create directory for solution**. You may deselect both **Add to source control** and **Show telemetry in the Windows Dev Center** if you don't wish to version your work or use Application Insights. Click **OK** to create the project.

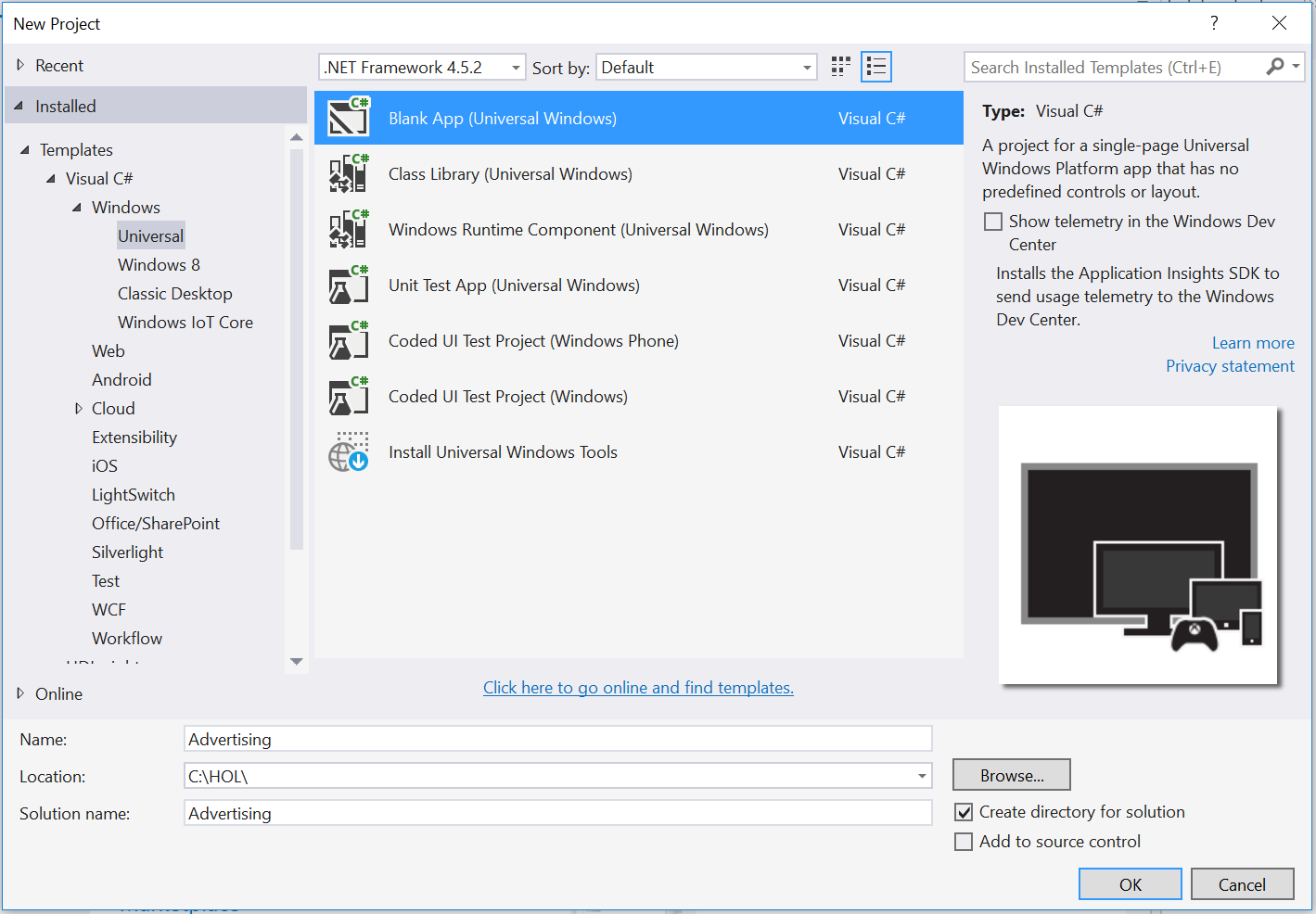


Figure 1

Create a new Blank App project in Visual Studio 2015.

1. Set your Solution Configuration to **Debug** and your Solution Platform to **x86**. Select **Local Machine** from the Debug Target dropdown menu.

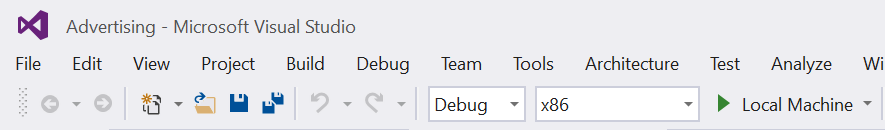


Figure 2

* + 1. Configure your app to run on the Local Machine.

1. Build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.

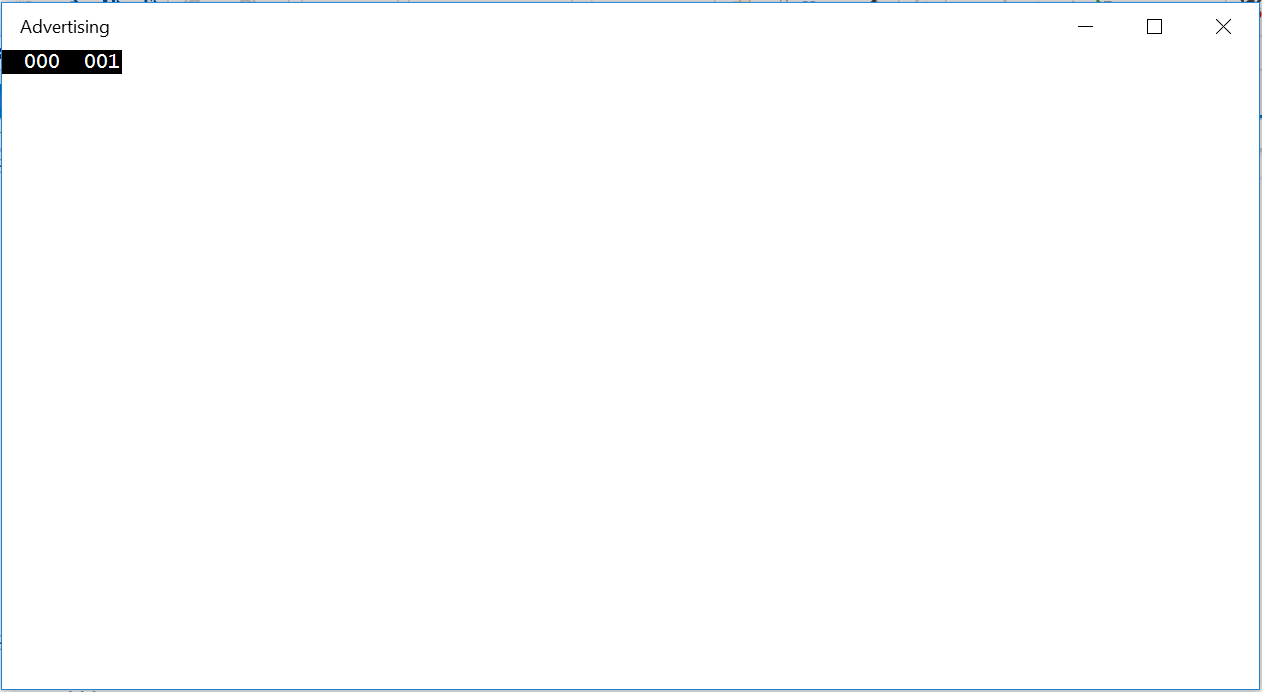


Figure 3

The blank universal app running in Desktop mode.

* 1. **Note:** The frame rate counter is a debug tool that helps to monitor the performance of your app. It is useful for apps that require intensive graphics processing but unnecessary for the simple apps you will be creating in the Hands-on Labs.
  2. In the Blank App template, the preprocessor directive to enable or disable the frame rate counter is in **App.xaml.cs**. The frame rate counter may overlap or hide your app content if you leave it on. For the purposes of the Hands-on Labs, you may turn it off by setting **this.DebugSettings.EnableFrameRateCounter** to **false**.

1. Return to Visual Studio and stop debugging.

Task 2 – Install the Windows 10 Advertising SDK

Before adding ads to your app, you will need to install the Windows Ad Mediator.

* 1. **Note:** The Ad Mediator includes the Microsoft Advertising SDK for XAML.

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.

In your browser, navigate to <https://visualstudiogallery.msdn.microsoft.com/401703a0-263e-4949-8f0f-738305d6ef4b>. Download the Ad Mediator by clicking the **Download** button.

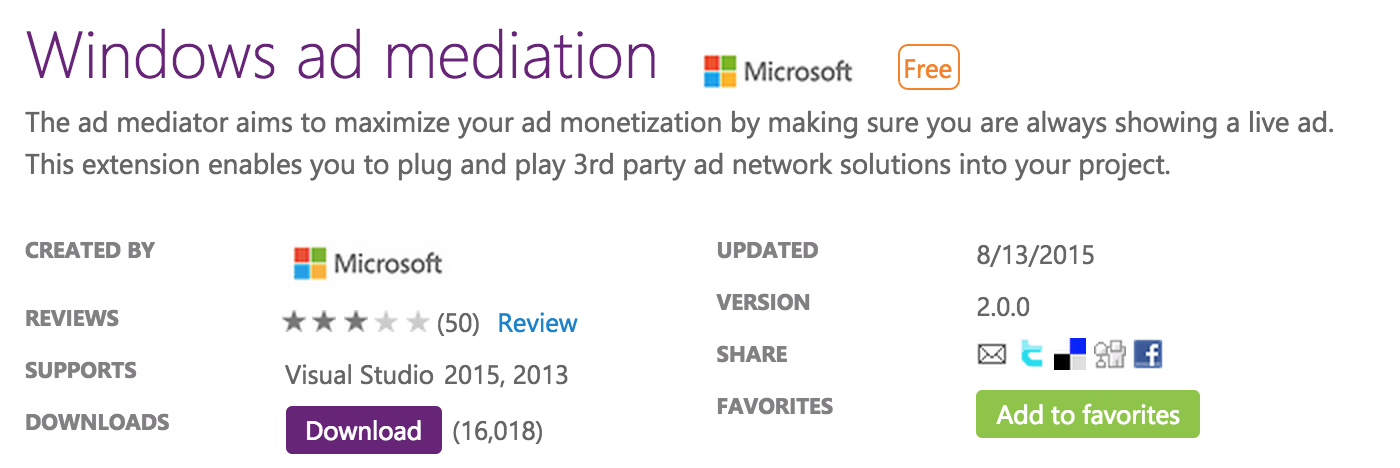


Figure 4

Download the Ad Mediator.

1. Run the msi file when prompted by the browser. When the Ad Mediator setup launches, install it using the default options. 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.

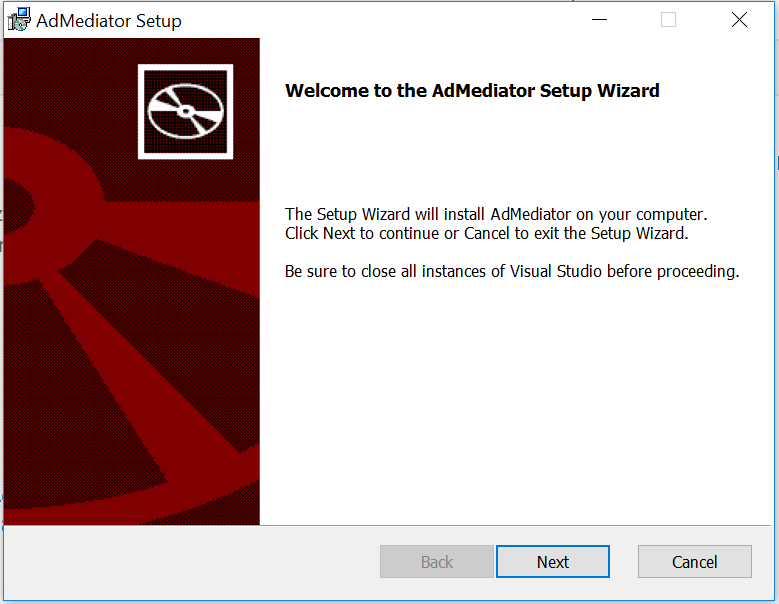


Figure 5

The Ad Mediator Setup Wizard..

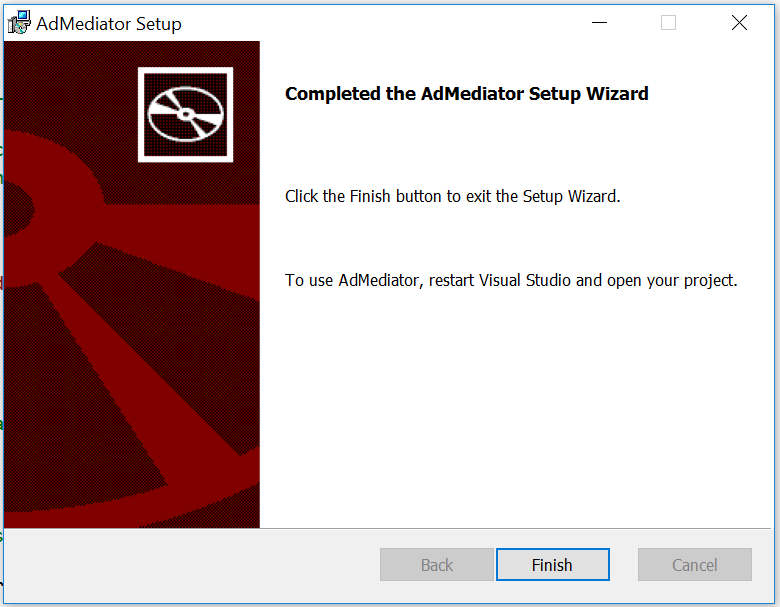


Figure 6

Complete the Ad Mediator installation.

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

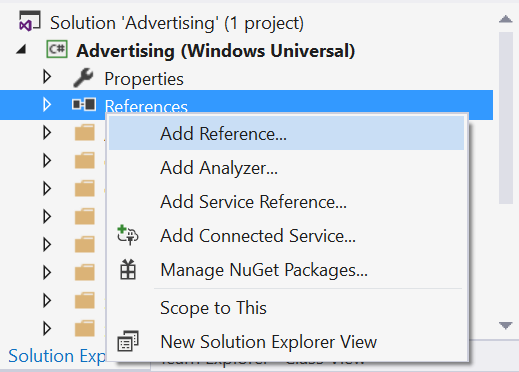


Figure 7

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.

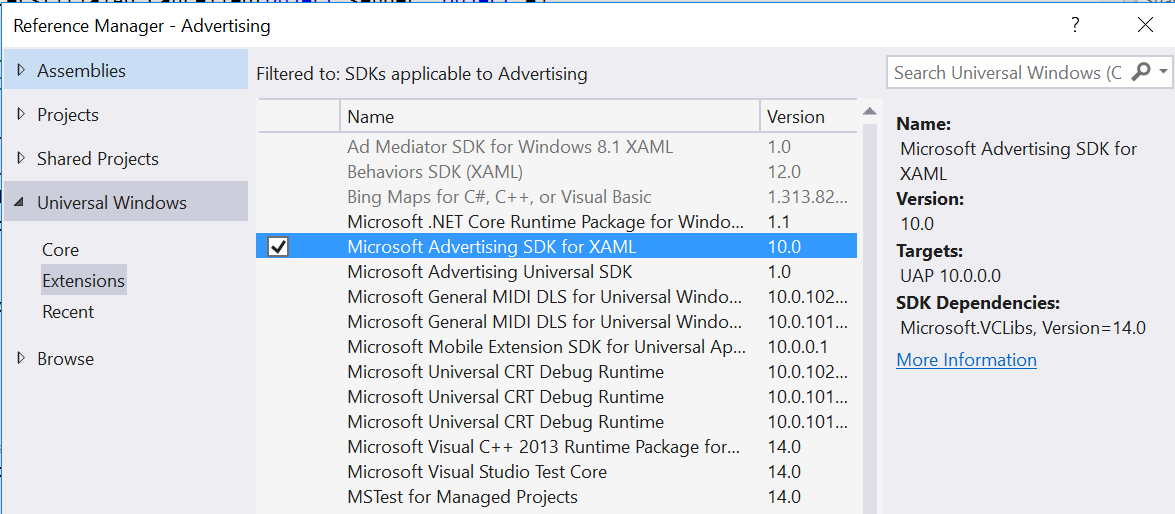


Figure 8

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.

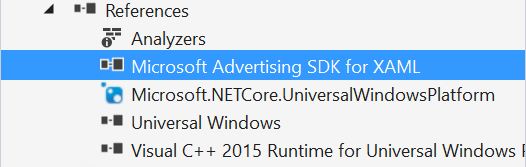


Figure 9

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. Right-click on the project name in the Solution Explorer and **Add > New Folder**. Name the folder **Models**.
2. To create the new DemoAds class, right-click on the **Models** folder 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**.

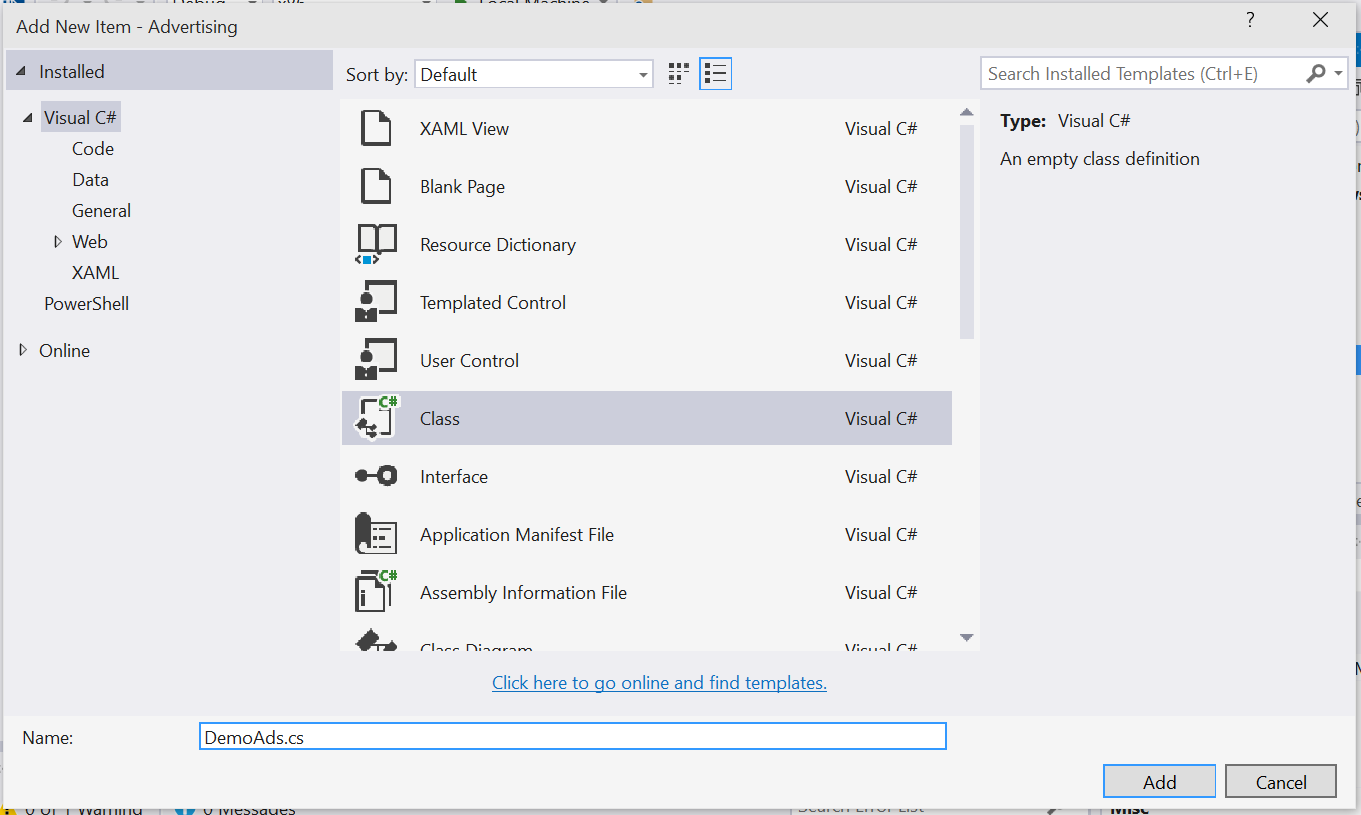


Figure 10

Create a new Visual C# class called DemoAds.

1. Open **DemoAds.cs**. 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 **MainPage.xaml.cs**. Add properties for **ShowAds** and **ViewedFullInterstitial**. You will require users to watch the full ad before continuing to the app.

C#

* 1. Public sealed partial class MainPage: Page
  2. {
  3. bool \_showAds = true;
  4. public bool ShowAds { get { return \_showAds; } set { \_showAds = value); } }
  5. bool \_viewedFullInterstitial = true;
  6. public bool ViewedFullInterstitial { get { return \_viewedFullInterstitial; } set { \_viewedFullInterstitial = value); } }

1. Add the **Microsoft.Advertising.WinRT.UI** and **Advertising.Models** namespaces to reference DemoAds and the Microsoft Advertising UI.

C#

* 1. using Microsoft.Advertising.WinRT.UI;
  2. using Advertising.Models;
  3. namespace Advertising
  4. {

1. After the class definition, add a private InterstitialAd field.

C#

* 1. namespace Advertising
  2. {
  3. public sealed partial class MainPage : Page
  4. {
  5. private InterstitialAd \_interstitialAd;
  6. bool \_showAds = true;

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();
  5. if (ShowAds)
  6. {
  7. // initialize the interstitial class
  8. \_interstitialAd = new InterstitialAd();
  9. // wire up all 4 events
  10. \_interstitialAd.AdReady += interstitialAd\_AdReady;
  11. \_interstitialAd.Cancelled += interstitialAd\_Cancelled;
  12. \_interstitialAd.Completed += interstitialAd\_Completed;
  13. \_interstitialAd.ErrorOccurred += interstitialAd\_ErrorOccurred;
  14. RequestAd();
  15. }
  16. else
  17. {
  18. // start normally
  19. }
  20. }

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. }

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. The interstitial video ad will play when the app loads.

Task 4 – Require the ad

* 1. Let’s require that the user watch an entire video ad before proceeding.

1. We will leverage the Cancelled event by displaying a message to the user and restarting the ad.

C#

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

* 1. {
  2. // raised when the user has watched the full video
  3. 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. For more on ContentDialogs, visit <https://msdn.microsoft.com/library/windows/apps/windows.ui.xaml.controls.contentdialog.aspx>

1. Build and run the app. Click the running video to display a back button. Select the back button while viewing the ad to see the behavior of the interstitialAd\_Cancelled event.
2. Stop debugging and return to Visual Studio.
3. 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. }

1. Build and run the app again. Confirm that you can skip the ad by selecting the back button while it is playing.
2. Stop debugging and return to Visual Studio.
3. 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 column in your app. The same techniques can be used to place ads inline within other layout controls.
4. Open **MainPage.xaml**. Add the following namespaces shown in red.

XAML

* 1. xmlns:UI="using:Microsoft.Advertising.WinRT.UI"

xmlns:c="using:Advertising.Converters"

1. In order to toggle visibility of the advertising section, we will use the **ValueWhenConverter**. Right-click on the project name in the Solution Explorer and choose **Add > New Folder**. Give it the name **Converters**.
2. Right-click on the **Converters** folder and choose **Add > Existing Item**. Browse to the Assets folder in your Hands-on lab directory and add the **ValueWhenConverter.cs** class. The ValueWhenConverter will allow you to associate a Boolean with the Visible/Collapsed properties on a XAML element.
3. Add a ResourceDictionary to MainPage.xaml to handle the conversion.

XAML

* 1. xmlns:c="using:Advertising.Converters">
  2. <Page.Resources>
  3. <ResourceDictionary>
  4. <c:ValueWhenConverter x:Key="VisibleWhenTrueConverter">
  5. <c:ValueWhenConverter.When>
  6. <x:Boolean>True</x:Boolean>
  7. </c:ValueWhenConverter.When>
  8. <c:ValueWhenConverter.Value>
  9. <Visibility>Visible</Visibility>
  10. </c:ValueWhenConverter.Value>
  11. <c:ValueWhenConverter.Otherwise>
  12. <Visibility>Collapsed</Visibility>
  13. </c:ValueWhenConverter.Otherwise>
  14. </c:ValueWhenConverter>
  15. </ResourceDictionary>
  16. </Page.Resources>

1. To create a container for your inline ad, replace the **Grid** in **MainPage.xaml** with a **Hub**. Add an **Advertising** hub section and an adjacent **Content** hub section to the hub. The Advertising 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. <Hub Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  2. <HubSection VerticalContentAlignment="Stretch" x:Name="AdvertisingSection"
  3. Header="Advertising"
  4. Visibility="{Binding Path=ShowAds, Converter={StaticResource ResourceKey=VisibleWhenTrueConverter}}">
  5. <DataTemplate>
  6. <Grid VerticalAlignment="Top">
  7. <!-- The rectangle acts as a place holder so we can see where the ad control is located-->
  8. <Rectangle Fill="Blue" Width="300" Height="600"/>
  9. <UI:AdControl ApplicationId="d25517cb-12d4-4699-8bdc-52040c712cab"
  10. AdUnitId="10043122"
  11. Height="600"
  12. VerticalAlignment="Top"
  13. Width="300"/>
  14. </Grid>
  15. </DataTemplate>
  16. </HubSection>
  17. <HubSection Header="Content" />
  18. </Hub>

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

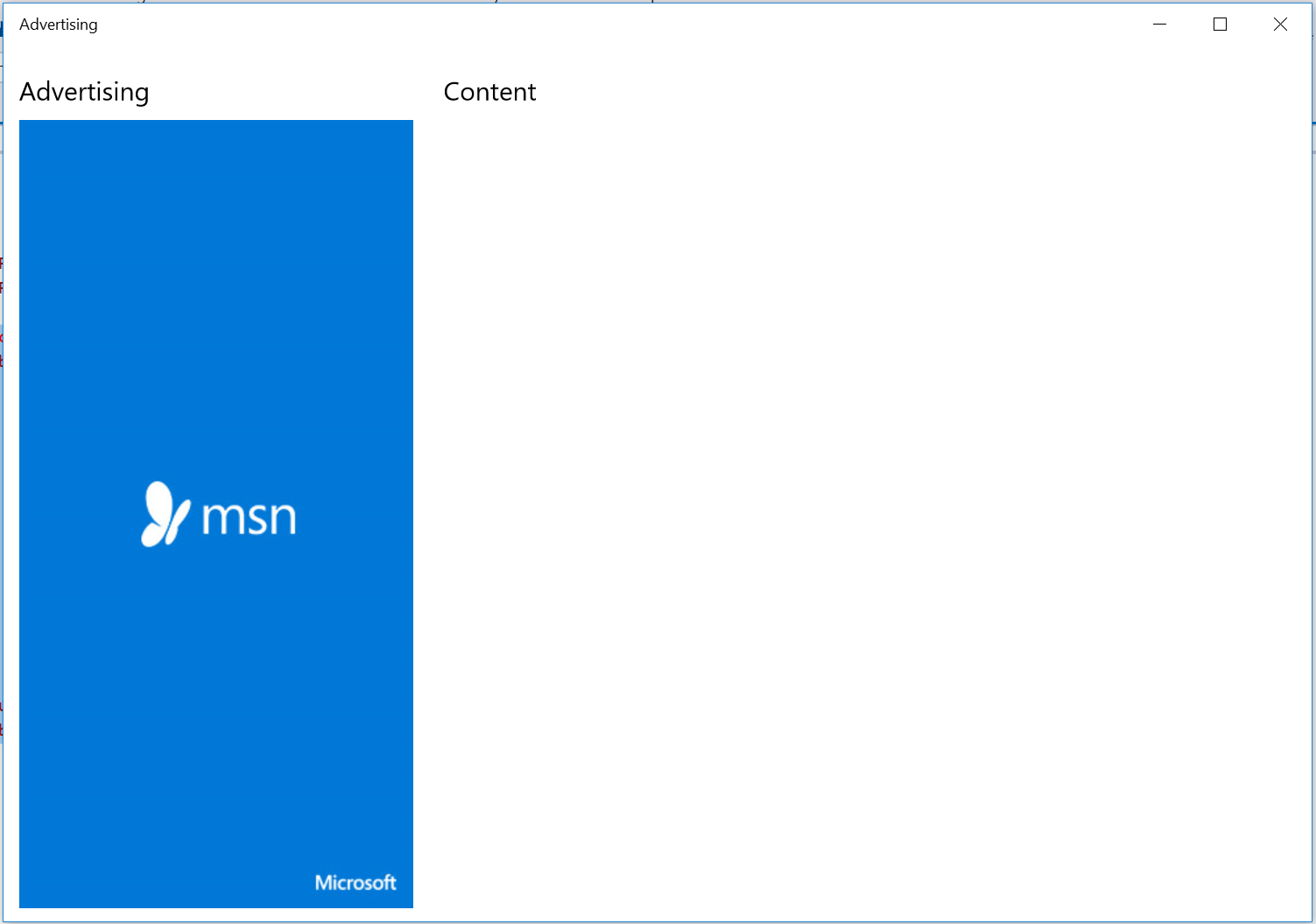


Figure 11

Inline ad in the Advertising app.

1. Stop debugging and return to Visual Studio.

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 a Hub control.