



## VMAX Windows Phone 8.1(XAML) SDK





# VMAX

*Windows Phone 8.1 (XAML)*



## [INTRODUCTION](#)

[What is this SDK?](#)

[Who should use this SDK?](#)

## [GETTING STARTED](#)

[Prerequisites](#)

[Obtaining the SDK](#)

[Setting up the SDK](#)

[Integrating the SDK](#)

## [REQUESTING ADS](#)

[Define a VMAXAdView with code](#)

[Requesting Interstitials](#)

[Requesting Rewarded Video Incent ads](#)

[Requesting Video Interstitial Ads for Video Player adspots](#)

[Requesting Banner Ads](#)

[Requesting Frame Ads](#)

[Requesting Native Ads](#)

## [OPTIONAL CALLBACKS](#)

[Methods](#)

[FailedToLoadAd Callback](#)

[FailedToCacheAd Callback](#)

[DidLoadAd Callback](#)

[DidCacheAd Callback](#)

[WillDismissOverlay Callback](#)

[WillLeaveApp Callback](#)

[WillPresentOverlay Callback](#)

## [TESTING INTEGRATION](#)

[Adding Test Devices](#)

[Using the Test Adspot Ids](#)

## [APPENDIX](#)

[A: Mediation](#)

[B: Native Ad Helper Library Usage](#)

[Introduction](#)

[Getting Started](#)

[Optional Callbacks](#)

[JSON Details](#)

[Customizing Native Ad](#)



## INTRODUCTION

This document provides technical guidance for using the VMAX SDK. Application developers can use this guide to integrate their applications with the VMAX SDK, specifying the types of Ads, they want to integrate.

This document also provides information on mediation integration with other supported Ad networks.

### **What is this SDK?**

The VMAX SDK is designed to help you integrate Ads within your application and fast track your way to monetizing your Windows Phone 8.1 (XAML) applications.

### **Who should use this SDK?**

This guide is made for Windows Phone 8.1 (XAML) App developers intending to include the VMAX WP 8.1 SDK into their Application to make Ad requests.



# GETTING STARTED

## Prerequisites

### Caution:

To utilize all features of the VMAX SDK and access all supported ad networks, please make sure that you have created a new Adspot.

An Adspot may be referred to as AdspotId in API calls.

Before beginning, ensure that:

1. You have a VMAX.com Developer Account. If not, sign up at our [Developer Portal](#).
2. Note your Adspot. You will require the Adspot when using the VMAX SDK to request Ads. Check for the default Adspot in your welcome email or visit <http://dashboard.vmax.mobi/ad-spot-list.html> to get a list of all your Adspot Ids.

## Obtaining the SDK

The first step to integrate the VMAX SDK with your WP 8.1(XAML) application is to download our SDK package from [here](#). Alternatively, you can obtain the SDK as a NuGet package using the following command in the Nuget Package Manager.

```
Install-Package com.vmax.windows.ads.wp81
```

## Setting up the SDK

Unzip the downloaded SDK Zip file into a temporary folder. Assuming that the folder is [VMAX\_SDK\_FOLDER], you will see the following contents:

<b>VMAXSDKWP81/packages</b>	This folder contains two folders - 1.) <b>VMAX</b> - contains VMAX SDK and its dependencies. 2.) <b>Mediation Partners</b> - Various mediation partners supported by VMAX SDK along with their mediators/adaptors.
<b>VMAXSampleAppWP81/</b>	Sample WP8.1 project source code that demonstrates VMAX SDK integration.(By default it has all the mediation partners integrated for demo purpose however you can select any or all of them if you need.)



## Integrating the SDK

**Note \*:-** If you are integrating this sdk in a Unity game for Windows phone 8.1 Store, refer the documentation specific to Unity Windows Phone 8.1. You can find it [here](#).

Complete the following steps to integrate the SDK with your application.

1. Include the **<packages>** folder in your project's folder structure.
2. Using **Project -> Add Reference -> Browse**, add the **com.vmax.windows.ads.wp81.dll** & **com.vmax.windowsPhone.ads.mediation.dll** to your project.

These dlls are present in the **packages/VMAX** folder.

**Note \*:-** If you are already having the older version of VMAX SDK(prior to A-WX-3.0.4) please remove the older references and follow these steps.(As it might cause problem with the old references.Replace the old packages/references with the new one.)

3. Ensure that your project has the following capabilities in the **appxmanifest** package:
  - i. **internetclientserver**
  - ii. **musiclibrary**
  - iii. **pictureslibrary**
  - iv. **videolibrary**
  - v. **appointments**
  - vi. **location**



## REQUESTING ADS

Windows Phone 8.1(XAML) apps are composed of UI elements. **VMAXAdView** is simply another element displaying ads that respond to user interaction..

To display an ad, you first create a **VMAXAdView** , then specify the **AdspotId** and **UX** type (**Banner/Interstitial/Frame/Native**) and finally request an ad.

Before starting using the SDK you have to import the namespaces for the same.  
You can refer to the VMAX namespace in your code by using either of the following directives.

### *Through code-behind in C#*

```
using com.vmax.windows.ads.wp81;  
using com.vmax.windows.ads;
```

### *Through XAML*

1. Import the declared namespace.
2. Copy the following statement to the start of your XAML page (where other XAML namespaces are declared) to:

```
xmlns:ad="using:com.vmax.windows.ads.wp81"
```

You can just define the **VMAXAdView** using the markup below directly in your XAML markup file.

```
<ad:VMAXAdView Height="100"  
  x:Name="adView"  
  AdspotId="MY_ADSPOT_ID"  
  UX="MY_AD_UX_TYPE"/>
```

where

"MY\_ADSPOT\_ID" has to be replaced with your actual VMAX AdSpot ID.

"MY\_AD\_UX\_TYPE" should be replaced with the **UX** type constant you wish to request.

**Note:** Banner, Interstitial, Frame and Native are currently supported.

### *Define a VMAXAdView with code*

You can also create the **VMAXAdView** in code. The following C# example constructs a banner ad by declaring a **VMAXAdView** and configuring it:

```
VMAXAdView adView=new VMAXAdView (); // To initialize the VMAXAdView  
  
// To set the mandatory properties and to configure the adView after initialization.  
  
adView.AdspotId = "MY_ADSPOT_ID"; // To specify the Adspot Id. Replace  
"MY_ADSPOT_ID" with your actual Adspot Id.
```





```
adView.UX= AdUX.Banner; // To specify banner ads.
```

You can also check the version of the sdk that you are using in case of any version specific support needed. VMAX sdk has an api **GetSDKVersion()**. It returns a string value that will give you the exact sdk version that has been integrated.

```
VMAXAdView adView=new VMAXAdView (); // To initialize the VMAXAdView  
var versionString = adView.GetSDKVersion();
```

## Requesting Interstitials

The VMAX SDK provides various mechanisms to request for Ads in your WP8.1 application. The SDK makes available both simple fully-managed Ad request methods and also the Cache & Show Ad methods to enable prefetching of ads for later viewing.

Managed Interstitials using LoadAd()

To request an interstitial, call the LoadAd() method.

For example:

```
VMAXAdView adView=new VMAXAdView (); // To initialize the VMAXAdView.  
  
adView.UX=AdUX.Interstitial; // To specify Interstitial ads.  
adView.AdspotId = "MY_ADSPOT_ID"; // To specify the Adspot Id. Replace  
"MY_ADSPOT_ID" with your actual Adspot Id.  
  
adView.TimeOut = 20; // To specify the timeout in case of Ad failure. Default is 20  
  
// Location and demographic targeting information may also be specified. Out of  
// respect for user privacy, VMAX asks that you only specify location and demographic  
// data if that information is already used by your app. Below are example of optional  
// properties:  
  
adView.UserInfo.Age ="age";  
adView.UserInfo.City = "city";  
adView.UserInfo.Country = "country";  
adView.UserInfo.DOB = "dob";  
adView.UserInfo.Email = "email";  
adView.UserInfo.Gender = "gender";  
  
Note* - where "age", "city", "country", "dob", "email" and "gender" should be  
replaced with the actual values meeting your targeting requirements.  
  
for e.g.-  
Note: Kindly don't copy paste the code as it is. As that may have bad impact on  
targeting of Ads served to user.  
  
adView.UserInfo.Age = 23;
```

```
adView.UserInfo.City = "Mumbai";
adView.UserInfo.Country = "India";
adView.UserInfo.Gender = eGender.Male;
adView.UserInfo.DOB = DateTime.Now;
adView.UserInfo.Email = "abc@xyz.com";

adView.LoadAd(); // This will load the Ad and on success provides a callBack named
DidLoadAd. You need to handle this to perform any action on this event in your
application.

//Interstitial Caching
//To Cache Interstitial Ads for later viewing is possible through the below API:
adView.CacheAd(); // To perform caching of the Ad. If required, this event can be
used to run any custom code.

//Showing Cached Ads:
adView.ShowAd(); // To show the Cached Ad

//Cancel Ads:
adView.CancelAd(); // To cancel the Cached Ad and free up all advview resources .
```

## Requesting Rewarded Video Incent ads

VMAX also supports video ads with incent provision. These are video ads which on view completion will provide some virtual currency to users.

Vmax SDK supports wallet management to maintain walletElements also known as currency name which stores respective currency amount or total reward amount. And provides provision of getting total virtual currency, spending virtual currency, awarding virtual currency.

Below snippet shows the example of requesting Vast MarkUp Ads

```
// Initialize the advview object with proper adspot id and ux type
VMAXAdView adView = new VMAXAdView();

adView.AdspotId = "MY_ADSPOT_ID";
adView.UX= AdUX.Interstitial;

//Initialize wallet object
Wallet wallet = Wallet.GetWallet();

//Create a wallet element
WalletElement element = wallet.AddWalletElement("SampleWalletPath", "CurrencyName");

//Rewarded Video
RewardedVideo rv = new RewardedVideo(element);

// Set the following flags of the rewarded video to true or false as per your need
to
```

```
// show different popups

    rv.showCappingPopup = true;
    rv.showInterruptedPopup = true;
    rv.showPrePopup = true;
    rv.showPostPopup = true;

// Set the texts for pre, post, interrupted and capping popups texts
    rv.PrePopupText = "Watch this video and get <reward> <currency>";
    rv.PrePopupTitle = "VMAX";
    rv.PostPopupText = "You earned <reward> <currency>";
    rv.PostPopupTitle = "VMAX";
    rv.InterruptedPopupText = "You will lose your reward";
    rv.InterruptedPopupTitle = "VMAX";
    rv.CappingPopupText = "Daily maximum count to view the ad reached";
    rv.CappingPopupTitle = "VMAX";

// Set the necessary event handlers
    element.OnUpdateFailedVirtualCurrency +=
Element_OnUpdateFailedVirtualCurrency;

    element.OnUpdateVirtualCurrency += Element_OnUpdateVirtualCurrency;
    rv.OnRewardedVideoCompleted += Rv_OnRewardedVideoCompleted;
    rv.OnRewardedVideoInterrupted += Rv_OnRewardedVideoInterrupted;
    rv.OnRewardedVideoPlaybackError += Rv_OnRewardedVideoPlaybackError;

// This is the most important API you need to call once initialization has been done
    adView.InitRewardedVideo(rv);

// Now you can call CacheAd() or LoadAd() according to your need.
    adView.LoadAd();

// If you need to award currency, use AwardVirtualCurrency API. It takes long value
as
// input parameter
    element.AwardVirtualCurrency(100);

// If you need to spend the currency, use SpendVirtualCurrency API. It takes long
// value as input parameter
    element.SpendVirtualCurrency(20);
```

```
// To get the currency for an WalletElement use GetVirtualCurrency
element.GetVirtualCurrency();
```

## Requesting Video Interstitial Ads for Video Player adspots

This will help you to get vast Markup(XML) in string format for video player Adspot where developers have their own players to play video Ads.

Below snippet shows the example of requesting Vast MarkUp Ads

```
// Initialize the adview object with proper adspot id and ux type and then call
// CacheAd() api.
VMAXAdView adView = new VMAXAdView();

adView.AdspotId = "MY_ADSPOT_ID";
adView.UX= AdUX.Interstitial;
adView.CacheAd();

//Once ad has been caached successfully, initialize the VastAdObject as follows.
private void AdView_AdCached(object sender, EventArgs e)
{
    VastAdObject adobj = adView.vastAdObject;
    if(adobj != null)
    {
        string str = adobj.GetVastResponse();
    }
}
```

## Requesting Banner Ads

Managed Banner Ads using LoadAd()

You can request for Banner Ads that refresh automatically in a container using the LoadAd() method. The signature for this method is:

```
VMAXAdView adView=new VMAXAdView (); // To initialize the VMAXAdView

//To set the mandatory properties and to configure the adView after initialization.
adView.AdspotId = "MY_ADSPOT_ID"; // To specify the Adspot Id. Replace
"MY_ADSPOT_ID" with your actual Adspot Id.
adView.UX= AdUX.Banner; // To specify banner ads.
```

```
// In case you are adding the control through codeBehind(C#) you have to place the  
VMAXAdView into a container control such as a Grid, StackPanel, and so on.  
this.ContentPanel.Children.Add(adView);
```

```
// Below are optional properties to enable / disable or modify refresh  
rate/stop/pause/resume. By default refresh is on in SDK with 30 sec
```

```
adView.RefreshRate = refreshRate; // To specify the refresh rate for banner ads  
Usage : int  
adView.Refresh =true; // To enable banner refresh - Usage: boolean  
adView.ResumeRefresh(); // To resume refresh  
adView.StopRefresh(); // To stop refresh  
adView.PauseRefresh(); // To pause refresh
```

```
Timeout and Demographics are same as above.
```

## Requesting Frame Ads

Pre-requisite for Frame Ads:

- Design below dimensions image frames:

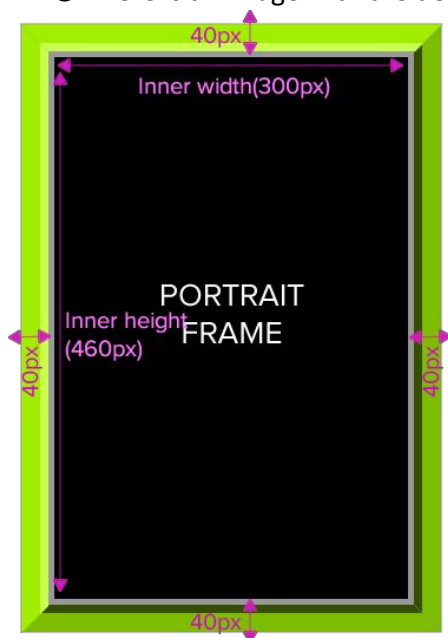
### Portrait:

Inner Dimensions	Outer Dimensions
300 X 460	380 X 540

### Landscape:

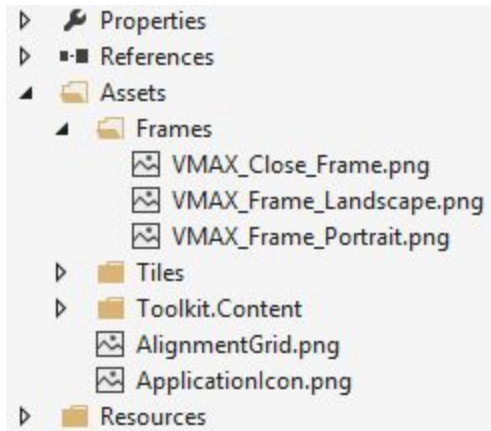
Inner Dimensions	Outer Dimensions
460 x 300	540 x 380

- Frame should be opaque inside with any color(Black recommended) upto inner dimensions mentioned above.
- Frame should be equally distributed outside 40 pixel each side so as to satisfy outer dimensions mentioned above
- That 40 pixel area can consists of Frame border each side. If in case, Frame border is less than 40 pixels thick then the remaining pixels should be set as transparent.
- Here is an image with the description:



- One Close button is required with below dimensions:
  - 36 x 36

- All above three assets (Two Frames [portrait and landscape] and One Close button) should be placed in a folder named **Frames** which needs to be created in the **Assets** folder of your Windows phone 8 Project. (Refer the sample app for the folder structure.)
- Here is an image of the solution explorer of Visual Studio Project specifying the exact place where you need to add the frames and the close button:



- Naming Conventions of above assets:
  - Landscape Frame image should be named as “VMAX\_Frame\_Landscape”. [extension like png]
  - Portrait Frame image should be named as “VMAX\_Frame\_Portrait”. [extension like png]
  - 36 x 36 Close button image should be named as “VMAX\_Close\_Frame”. [extension like png]

To request a Frame Ads, Follow below example for reference:

```
//Follow the same code snippet as mentioned in Requesting Interstitials with only
one change. //set UX property as frame instead of Billboard.

VMAXAdView adView=new VMAXAdView (); // To initialize the VMAXAdView

adView.UX = AdUX.Frame; //This will set Frame type explicitly.
```

## Requesting Native Ads

Native ads are ads which you can customize to match the unique user experience of your app. Native ads can be requested in the following manner -

```
VMAXAdView adView=new VMAXAdView (); // To initialize the VMAXAdView
```

```
// To set the mandatory properties and to configure the adView after initialization.
adView.AdspotId = "MY_ADSPOT_ID"; // To specify the Adspot Id. Replace
"MY_ADSPOT_ID" with your actual Adspot Id.
adView.UX= AdUX.Native; // To specify native ads.
```

```
adView.DidCacheAd +=AdView_AdCached; // add handler for successful callback of
caching of native ad.
adView.FailedToCacheAd +=DidFailed_CacheAd; // add handler for failure callback of
caching of native ad.
```

To request Native ads -

```
adView.CacheAd();
```

**Note :** `LoadAd()` is not allowed for requesting Native ads. Using that will give an error message.

The **VMAXAdView** provides the success and failure callback for Caching of Ads. The success callback can be used to provide the customization logic of the ad Unit.

**NativeAd** is represented by the **NativeAd** property of the **VMAXAdView** object that you created which gives you an object of type **NativeAd**. In case of success it will contain the various elements of the **NativeAd**.

You can extract the values of **NativeAd** elements by using the **GetElementValue(<Predefined constant values>)** API of **NativeAd** class.

- Predefined Constants are defined in the class- **NativeAdConstants**.

or

If you want all the elements in the form of a single **JSON** object and provide your own parsing logic, you can use the **GetContent()** API of the **NativeAd** class, which will return the **NativeAd** content in the form of JSON.

eg.

```
private void AdView_AdCached(object sender, EventArgs e)
{
    var nativeVMAXAdView = sender as VMAXAdView;
    if (nativeVMAXAdView.NativeAd!=null)
    {
        var nativeAd = nativeVMAXAdView.NativeAd; // gets the NativeAd property of
the VMAXAdView class.
        var nativeAdCustomJson = nativeAd.GetContent(); // gets the Native Ad
content as a JSON object.You can provide your own custom logic to parse this or
fetch values of individual elements as shown below.

        var buttonLabel = nativeAd.GetElementValue (NativeAdConstants.BUTTON_LABEL) ;
        var titleText = nativeAd.GetElementValue (NativeAdConstants.TITLE_TEXT) ;
        var imageUrl = nativeAd.GetElementValue (NativeAdConstants.LARGE_IMAGE) ;
        var iconUrl = nativeAd.GetElementValue (NativeAdConstants.ICON) ;
        var description = nativeAd.GetElementValue (NativeAdConstants.DESRIPTION) ;
    }
}
```

**Note:** On Failure of Cache request the **NativeAd** property of **VMAXAdView** object will have the value **null**.



### Impression and Click Logging-

Impression and Click logging will be done by the application itself. For this, just use the following API -

```
NativeAdHelper.RegisterViewForInteraction(adView, NativeAdGrid,  
listOfNativeClickableElements );
```

`listOfNativeAdClickableElements` is nothing but a list of elements in the rendered native ad which you want to make clickable. To make the entire native ad clickable you can pass this parameter as null. It can be initialised as below:

```
List<UIElement> listOfNativeClickableElements = new List<UIElement>();  
listOfNativeClickableElements.Add(description);  
listOfNativeClickableElements.Add(this);
```

Remember: Only after registering the impression the ad will be considered successfully rendered in application and impression will be recorded. Also, Only after registering the `nativeAdView` for clicks the clicks on the ad will be recorded.

Absence of calling this mandatory API on successful ad rendering will impact revenue so just ensure that you make a call to it as soon as the `NativeAd` gets loaded.

The SDK will automatically prevent a duplicate impression from being recorded for a single request.

**Note :** In cases where you re-use the view to show different ads over time, make sure to call `UnRegisterNativeViewForCallToAction()` before registering the same view with a different instance of `NativeAd`.

VMAX also provides a helper library to render the native ad. [This](#) Appendix section will summarize the use of Native Ad helper library.



## Requesting Video Interstitial Ads for Video Player adspots

This will help you to get vast Markup(XML) in string format for video player Adspot where developers have their own players to play video Ads.

Below snippet shows the example of requesting Vast MarkUp Ads

```
// Initialize the adview object with proper adspot id and ux type and then call
// CacheAd() api.
VMAXAdView adView = new VMAXAdView();

adView.AdspotId = "MY_ADSPOT_ID";
adView.UX= AdUX.Interstitial;
adView.CacheAd();

//Once ad has been caached successfully, initialize the VastAdObject as follows.
private void AdView_AdCached(object sender, EventArgs e)
{
    VastAdObject adobj = adView.vastAdObject;
    if(adobj != null)
    {
        string str = adobj.GetVastResponse();
    }
}
```

## OPTIONAL CALLBACKS

The VMAXAdView generates callbacks in following scenarios:

- Successful and unsuccessful fetching attempts of Ads

### Methods

Specifying callbacks in XAML-

```
<ad:VMAXAdView
    x:Name="adView"
    DidCacheAd="AdCached"
    DidLoadAd="AdReceived"
    FailedToLoadAd="AdFailed"
    WillDismissOverlay="AdCollapsed"
    WillLeaveApp="LeavingApplication"
```

```
WillPresentOverlay="AdExpanded"  
FailedToCacheAd="DidFailed_CacheAd"/>
```

Specifying callbacks through Code-

```
adView.DidLoadAd+=AdReceived;  
adView.FailedToLoadAd+=AdFailed;  
adView.WillDismissOverlay+=AdCollapsed;  
adView.WillLeaveApp+=LeavingApplication;  
adView.WillPresentOverlay+=AdExpanded;  
adView.FailedToCacheAd+=DidFailed_CacheAd;
```

The following are different methods to generate an API callback.

### ***FailedToLoadAd Callback***

The VMAX SDK offers this callback, in cases where an Ad is not served. If required, this event can be used to run any custom code.

```
private void AdFailed(object sender,  
com.vmax.windows.ads.wp81.VMAXAdView.AdFailedEventArgs e)  
{  
    // ..Custom code goes here  
}
```

### ***FailedToCacheAd Callback***

The VMAX SDK offers this callback, in cases where an Ad is not cached. If required, this event can be used to run any custom code.

```
private void DidFailed_CacheAd(object sender, EventArgs e)  
{  
    // ... Custom code goes here  
}
```

### ***DidLoadAd Callback***

The VMAX SDK offers this callback, in cases where an Ad is rendered successfully. If required, this event can be used to run any custom code.

```
private void AdReceived(object sender, EventArgs e)  
{  
    // ..Custom code goes here  
}
```

### ***DidCacheAd Callback***

The VMAX SDK offers this callback, in cases where an Ad is cached successfully. If required, this event can be used to run any custom code.

```
private void AdCached(object sender, EventArgs e)
{
    // ..Custom code goes here
}
```

### ***WillDismissOverlay Callback***

The VMAX SDK offers this callback, in cases of an Ad dismissal. If required, this event can be used to run any custom code.

```
private void AdCollapsed(object sender, EventArgs e)
{
    // ..Custom code goes here
}
```

### ***WillLeaveApp Callback***

The VMAX SDK offers this callback, in cases of the Ad closing the current App and opening another App. If required, this event can be used to run any custom code.

```
private void LeavingApplication(object sender, EventArgs e)
{
    // ..Custom code goes here
}
```

### ***WillPresentOverlay Callback***

The VMAX SDK offers this callback, in cases where an interstitial/overlay is shown . If required, this event can be used to run any custom code.

```
private void AdExpanded(object sender, EventArgs e)
{
    // ..Custom code goes here
}
```



## TESTING INTEGRATION

You can test the SDK integration using any of the following methods:

- Set your integration to Test mode for certain devices.
- Set your integration to use Test Adspot Ids, which VMAX provides.

### Adding Test Devices

The VMAX SDK allows you to set a number of devices as Test Devices by passing their AdvertisingID to the SDK. You can add test devices by using the TestDevice property which takes a list of id (string) as its parameters.

```
adView.TestDevice = new List<string>() { [your advertising id/ids here] };
```

Each Device's AdvertisingID can be fetched by using the following piece of code in your application.

```
// Place this snippet in a method and take the value returned by it and use it as device id.
try
{
    var advertisingIdToPrint =
Windows.System.UserProfile.AdvertisingManager.AdvertisingId;
}
catch { }
```

Once you have this list of device ids, this is how you can enable test mode.

The sdk exposes an API by name **SetTestDevices()** with a couple of parameters as:

**SetTestDevices(VMAXAdView.TestMode testMode, List<string> listOfStringsOfDeviceIDs)**

**VMAXAdView.TestMode:** This parameter will allow you to set the test mode that you want to set. It can have any of these values.

**TEST\_VIA\_ADVID:** this mode will need the list of device ids that is the second parameter. Test mode will be enabled for only those devices.

**TEST\_FOR\_ALL\_DEVICES:** This mode will not require the list in the second parameter. You can pass a null there. All the devices will show test ads if this mode has been selected.

**TEST\_VIA\_ID\_FROM\_NETWORKS:** Reserved for future use.

Using this api you can set the test mode for your list of devices as shown below.

```
adView.SetTestDevices(VMAXAdView.TestMode.TEST_FOR_ALL_DEVICES, null);
//or
adView.SetTestDevices(VMAXAdView.TestMode.TEST_VIA_ADVID, listDeviceIds);
```



## Using the Test Adspot Ids

You can also test your integration across our solutions by using the following Adspot Ids

Billboard Test Adspot Id : **8063**

Banner Test Adspot Id: **20846**

Video player Adspot Id: **V762947E3**

Native Infeed Test Adspot Id: **010116D2**

Native Content Stream Test Adspot Id: **V76062644**

**Note:** Please make sure your production App does not utilize these Adspot Ids as you would not be earning revenue using these Adspot Ids.

## APPENDIX

### A: Mediation

To support Mediation with partners:

1. The “**Mediation Partners**” folder inside the “**VMAXSDKWP81/packages**” folder consists of all supported Client to Server integration partners like Microsoft, AdDuplex and their respective adapters/mediator dlls.
2. Browse and add references to the dlls/Extensions in your project that are present in the corresponding folder.(refer to the below table)  
For example- for mediating through **Microsoft**, add the reference of the dlls to your project which are present in the “**Microsoft**” folder (**MicrosoftMediator\_WP81.dll**) & **Microsoft Advertising SDK for Windows Phone 8.1 (XAML) Extension**.
3. You can add/remove the dlls/Extensions based on your choice of the mediation partners that you want. Just make sure to add/remove all the dlls/Extensions for a specific partner that are present in the respective folder.(This will help you in reducing the unwanted size overhead)
4. Fill required details on the VMAX web interface at [mp.vserv.com/ad-spot-list.html](http://mp.vserv.com/ad-spot-list.html) to start mediation for that Adspot Id.
5. The following table describes the files to be added for the respective mediation partners -

Partner	Folder where respective mediation files placed(inside VMAXSDKWP81/packages/ Mediation Partners)	.dll/Extensions to be selected and added as a reference to the Project.
AdDuplex	AdDuplex	<a href="#">AdDuplex SDK for Windows Phone 8.1 (XAML) (Extension)</a> AdDuplexMediator_WP81.dll
Microsoft	Microsoft	<a href="#">Microsoft Advertising SDK for Windows Phone 8.1 (XAML) (Extension)</a> MicrosoftMediator_WP81.dll

**Note :-** To install “**Microsoft Advertising SDK for XAML**” & “**AdDuplex SDK for Universal Apps**” Extensions please visit the hyperlinks mentioned in the table and perform the steps necessary to install the SDKs.

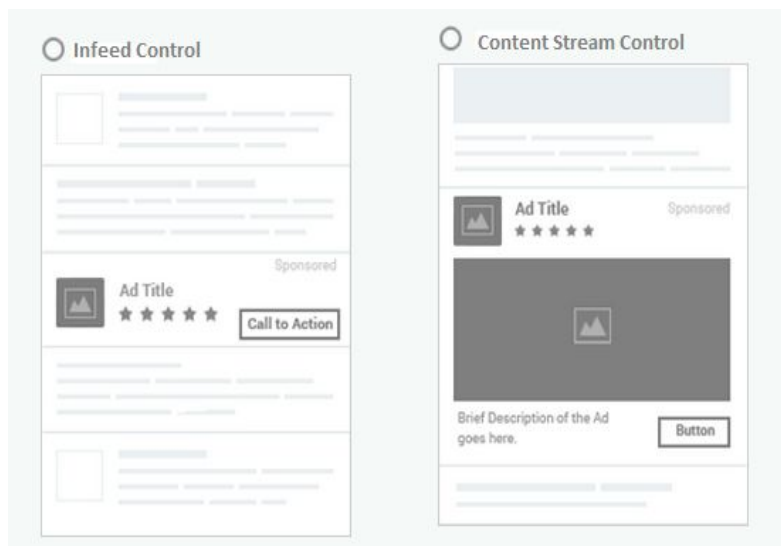
**Note :-** All the capabilities required for mediation partners are included in the section “[Integrating The SDK](#)” mentioned above. Make sure you have included all of them.

## B: Native Ad Helper Library Usage

### Introduction

This document will help you utilize a Native Ad Helper Library to render the native ads in a couple of predefined formats.

The VMAX Native Ad Helper Library has been designed to reduce the efforts of the developer in rendering the native ads. The library helps in rendering the native ads in any one of the two formats. **Infeed** and **Content Stream**. Here is an image showing the actual format.



### Integrating the SDK

Complete the following steps to integrate the helper library with your application.

1. Include the <packages> folder in your project's folder structure if you haven't done so yet.
2. Using Project -> Add Reference -> Browse , add the VMAXNativeAdHelper.WP81.dll to your project. It is present in the packages/VMAX folder.



## Getting Started

The library internally provides two controls which can be added to the xaml file. **InfeedControl** and **ContentStreamControl**. You may even create an object of these control in the backend and add it to any one of your grids programmatically. This is how the library can be referred in the xaml

```
xmlns:VMAXControl="using:VMAXNativeAdHelper.WP81"
```

and this is how the controls can be referred to.

```
<Grid x:Name="MainGrid">
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <VMAXControl:InFeedControl Name="InfeedAdControl"
                               Grid.Row="0"
                               FailedToRenderNativeAd="InfeedAdControl_FailedToRenderNativeAd"
                               DidRenderNativeAd="InfeedAdControl_DidRenderNativeAd"
                               Height="Auto"/>

    <VMAXControl:ContentStreamControl Name="ContentStreamAdControl"
                                      Grid.Row="1"
                                      FailedToRenderNativeAd="ContentStreamAdControl_FailedToRenderNativeAd"
                                      DidRenderNativeAd="ContentStreamAdControl_DidRenderNativeAd"
                                      Height="Auto"/>
</Grid>
```

For code behind access, add the namespace after referring the dll in the project as:

```
using VMAXNativeAdHelper.WP81;
```

Once the controls have been placed in the xaml, what remains is the actual native ad data to be shown in these controls. You can get the native ad data using the VMAX Windows phone sdk by selecting appropriate Ad UX Types and using proper Adspot Id as shown [here](#). You should, then, call an API **SetContentAndShowNativeAd()** on the controls. This API takes JObject (JSON object) as a parameter. (If you do not have a reference to the **Newtonsoft.Json** library, add it using the nuget package manager.) This JObject, you can obtain from the **NativeAd** property of the adView. The **GetContent()** API will return you a JObject from the native ad as shown here.

```
InfeedAdControl.SetContentAndShowNativeAd(adView.NativeAd.GetContent());
ContentAdControl.SetContentAndShowNativeAd(adView.NativeAd.GetContent());
```

Once you call this API with its necessary parameter, the control will be populated with the necessary parameters from the JObject that was provided.

Here is a sample image showing the appearance of the ads:

Infeed Ad



Content Stream Ad



## Optional Callbacks

### 1. DidRenderNativeAd()

This callback can notify you about the successful rendering of the native ad.

### 2. FailedToRenderNativeAd()

You can register for an event **FailedToRenderNativeAd** which can tell you the reason behind the failure in rendering the native ad. One of the reasons can be Empty or Improper JObject being passed to the **SetContentAndShowNativeAd()** API.

```
private async void InfeedAdControl_FailedToRenderNativeAd(NativeAdEventArgs e)
{
    await new MessageDialog(e.ErrorMessage, "Infeed Ad Control").ShowAsync();
}
```

```
private async void InfeedAdControl_DidRenderNativeAd()
{
    await new MessageDialog("Infeed ad shown successfully", "Infeed Ad
Control").ShowAsync();
}

private async void ContentStreamAdControl_FailedToRenderNativeAd(NativeEventArgs
e)
{
    await new MessageDialog(e.ErrorMessage, "Content Stream Ad
Control").ShowAsync();
}

private async void ContentStreamAdControl_DidRenderNativeAd()
{
    await new MessageDialog("Content Stream ad shown successfully", "Content Stream
Ad Control").ShowAsync();
}
```

The class **NativeEventArgs** contains a field by name **ErrorMessage** which can tell you the exact reason for the failure in rendering the native ad.

## JSON Details

The VMAX Server gives you a JSON that will appear as below:

```
{
  "BUTTON_LABEL": "Install Now",
  "ICON_48x48": "sample http image url",
  "ICON_80x80": "sample http image url",
  "SCREENSHOT_300x250": "sample http image url",
  "SPAN_320x50": "sample value",
  "LARGE_IMAGE_1200x627": "sample http image url",
  "TITLE_TEXT": "Lost Twins - A Surreal Puzz...",
  "DESCRIPTION": "Lost Twins is a delightful indie game that is original, beautifully made...",
  "PRICE": "Free",
  "TOTAL_INSTALLS": "100",
  "REVIEW": "sample string",
  "RATING": "4.1",
  "SIZE": "2MB",
  "TOTAL_REVIEWS": "1100",
  "PRODUCT_CATEGORY": "Action Games",
  "IMPRESSION_URL": "sample http impression url",
  "CLICK_URL": "sample http click url"
}
```

It has several fields out which help in rendering the native ad depending upon the type. The VMAX Sdk also internally gets the same JSON. But it adds more elements to it and creates a new JSON like:

```
{
  "BUTTON_LABEL": "Install Now",
  "ICON": "sample http image url",
  "ICON_HEIGHT": "48",
  "ICON_WIDTH": "48",
  "SCREENSHOT": "sample http image url",
  "SCREENSHOT_HEIGHT": "800",
  "SCREENSHOT_WIDTH": "480",
  "SPAN": "sample http image url",
  "SPAN_HEIGHT": "480",
  "SPAN_WIDTH": "800",
  "LARGE_IMAGE": "sample http image url",
  "LARGE_IMAGE_HEIGHT": "1200",
  "LARGE_IMAGE_WIDTH": "627",
  "TITLE_TEXT": "Lost Twins - A Surreal Puzz...",
  "DESCRIPTION": "Lost Twins is a delightful indie game that is original, beautifully...",
  "PRICE": "Free",
  "TOTAL_INSTALLS": "100",
  "REVIEW": "sample string",
  "RATING": "4.1",
  "SIZE": "2MB",
  "TOTAL_REVIEWS": "1100",
  "PRODUCT_CATEGORY": "Action Games",
  "IMPRESSION_URL": "sample http impression url",
  "CLICK_URL": "sample http click url"
}
```

You can pass any of the above mentioned forms of JSON objects (in a JObject format) to the **SetContentAndRenderNativeAd()** API and the ad will be rendered according to the type of control used (Infeed or Content Stream).

## Customizing Native Ad

If you wish to customize the controls in their appearance, the following is the list of attributes which you can change:

Properties	Can developer modify
Background Color	yes
Foreground Color	yes
Border Thickness	yes
Border Brush	yes
Font Family	yes
Font Style	yes
Font Weight	no

Font Stretch	yes
Margin	no (Changes in this attribute may distort the control)
Width	no (Changes in this attribute may distort the control)
Height	no (Changes in this attribute may distort the control)
Horizontal Alignment	yes
Vertical Alignment	yes
Changes in Phone Theme Color	yes

The stars in the rating control take up the color of the theme applied.

**\*Note- Make sure you add the controls in a grid or any container with Height="Auto" as shown in the snippets above.**