**Advertising Overview**

There are two different paths you can choose for supporting advertising in the Silverlight Media Framework.

*Simple linear advertising (introduced in SMF 2.0)*

SMF 2.0 introduced an API to support simple linear advertising for players using the smooth streaming plugin. The idea was to provide an API to make it easy to play video ads as clips within the player. You can do this by assigning Advertisement object(s) to the PlaylistItem’s .PreRollAdvertisement, .InterstitialAdvertisements, & .PostRollAdvertisement properties. The Advertisement object contains its own properties to instruct the player when to schedule the ad, what ad to play, and how long to play the ad. You can do this in Xaml (see below) or in code via the SMFPlayer.ScheduleAdvertisement method.

<Media:PlaylistItem.InterstitialAdvertisements>

    <Media:Advertisement AdSource="http://smf.vertigo.com/videos/wildlife.wmv" StartTime="0:0:05" Duration="0:0:05" DeliveryMethod="ProgressiveDownload" />

</Media:PlaylistItem.InterstitialAdvertisements>

*Standards based advertising (introduced in SMF 2.5)*

SMF 2.5 introduces a new API and set of plugins to support industry standard advertising specifications such as MAST, VAST, & VPAID. This makes it very easy to integrate advertising support into your application and connect to ad servers that support these standards. The focus of the rest of this document is to explain how to use this new advertising feature.

**The big questions: when, where, what, & how**

**When** should your ads play? This is also known as **sequencing**. **MAST** is the standard that makes it easy to define this in XML and trigger your ads based on conditions.

**Where** should your ads appear in your application? This is also known as **targeting**. Besides sequencing, **MAST** also defines a standard for describing where to play your ads.

**What** ads should play? We refer to this as **handling** an ad. **VAST** is the standard that makes it easy to describe this in XML and is supported by most modern ad servers today. VAST additionally describes how those ads should be tracked.

**How** do you actually play an ad? **Ad players** are used to play ads of various types (video, images, Silverlight and Flash). **VPAID** provides a common interface that makes it easy to build ad players for any media type.

**Sequencing**

The very first step to adding ad support is determining **when** to play your ads. This concept (also known as “sequencing”), can be implemented in one of two ways: You can either use MAST, or you can build your own custom sequencing engine.

*MAST (Media Abstract Sequencing Template)*

MAST is a standard initiated by Akamai in order to define a common approach to address ad sequencing and resolve the issue of having different sequencing implementations for every deployment. Specifically, MAST defines an XML schema that can be used to describe the sequencing rules and targets of your ads. In other words, this describes both **when** and **where** to play your ads.

SMF now ships with a plugin that you can use to load MAST XML files and trigger ads to occur. To enable this feature:

* Add the assembly: **Microsoft.SilverlightMediaFramework.Plugins.Advertising** to your project’s references.
* Set the MarkerResources property on your PlaylistItem either in code or in Xaml as shown below:

<smf\_media:PlaylistItem …etc…>

    <smf\_media:PlaylistItem.MarkerResources>

        <smf\_media:MarkerResource Format="IAB-MAST" Source="mast-example.xml" />

    </smf\_media:PlaylistItem.MarkerResources>

</smf\_media:PlaylistItem>

Note: in place of mast-example.xml, put the Url of your MAST file. This is usually a remote Url.

**MAST polling:** You can also configure the marker resource to poll your MAST file at regular intervals by setting MarkerResource.PollingInterval. When a PollingInterval is specified, MAST triggers will be merged by ID. Triggers missing from the newer MAST document will be removed, triggers with matching IDs will be updated, and new triggers will be added.

*The MAST schema*

Let’s take a look at what a MAST file looks like. For more information about the MAST schema please refer to <http://openvideoplayer.sourceforge.net/mast/mast_specification.pdf>

Here is an extremely basic MAST file describing one preroll and one midroll with a description of the primary parts at the bottom.

<?xml version="1.0" encoding="utf-8" ?>

<MAST xsi:schemaLocation="http://openvideoplayer.sf.net/mast <http://openvideoplayer.sf.net/mast/mast.xsd>" xmlns="http://openvideoplayer.sf.net/mast" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <triggers>

    <trigger id="preroll" description="preroll"  >

      <startConditions>

        <condition type="event" name="OnItemStart" />

      </startConditions>

      <endConditions>

        <condition type="event" name="OnItemEnd"/>

      </endConditions>

      <sources>

        <source uri="/ClientBin/vast2-example1.xml" format="vast">

          <sources />

          <targets />

        </source>

      </sources>

    </trigger>

    <trigger id="midroll" description="midroll at 30 sec."  >

      <startConditions>

        <condition type="property" name="Position" value="00:00:15.0" operator="GEQ" />

      </startConditions>

      <endConditions>

        <condition type="event" name="OnItemEnd"/>

      </endConditions>

      <sources>

        <source uri="/ClientBin/vast2-example2.xml" format="vast">

          <sources />

          <targets />

        </source>

      </sources>

    </trigger>

  </triggers>

</MAST>

At its most basic level, a MAST document is composed of a collection of triggers. Each trigger can cause one or more ad to play.

**StartConditions:** This contains one or more conditions used to determine when your ad should begin. MAST allows for very complex conditions by allowing a condition to monitor an event or property of the player. Additionally, you can use multiple conditions to imply an OR relationship, or use nested conditions to imply an AND relationship.

**EndConditions:** Ads typically have their own duration and determine when they should end so while EndConditions can be used to determine when to stop an active ad; it is not its primary use. In addition to stopping an active ad, when an ad’s end condition is met, it recycles the trigger so it can be invoked again. In other words, the StartConditions can only be satisfied again once the EndConditions are met.

**Event Conditions:** There are a number of events you can monitor in your conditions.

|  |  |
| --- | --- |
| OnPlay | Defined as anytime the play command is issued, even after a pause |
| OnStop | The stop command is given |
| OnPause | The pause command is given |
| OnMute | The player was muted, or volume brought to 0 |
| OnVolumeChange | Volume was changed |
| OnEnd | The player has stopped naturally, with no new content |
| OnSeek | The player was manually seeked |
| OnItemStart | A new item is being started |
| OnItemEnd | The current item is coming to the end |
| OnFullScreenChange | Fullscreen has been toggled |
| OnPlayerSizeChanged | Player size has changed |
| OnError | An error has occurred, typically of enough severity to warrant display to the user |
| OnMouseOver | The mouse has moved |

**Property conditions:**

|  |  |  |
| --- | --- | --- |
| TimeSpan | Duration | The duration of the current content |
| TimeSpan | Position | The position of the current content |
| TimeSpan | WatchedTime | The amount of time that this item has rendered, regardless of seeks |
| TimeSpan | TotalWatchedTime | The total amount of content that has been rendered in this session |
| DateTime | SystemTime | The current system time |
| bool | FullScreen | True if the player is fullscreen |
| bool | IsPlaying | True if the player is playing content |
| bool | IsPaused | True if the player is paused |
| bool | IsStopped | True if the player is stopped, or not yet started |
| bool | CaptionsActive | True if captions are active and being shown |
| bool | HasVideo | True if the current content has a video stream |
| bool | HasAudio | True if the current content has an audio stream |
| bool | HasCaptions | True if the current content has captions available |
| int | ItemsPlayed | The count of items that have been displayed in full or part. |
| int | PlayerWidth | The physical width of the player application |
| int | PlayerHeight | The physical height of the player application |
| int | ContentWidth | The native width of the current content |
| int | ContentHeight | The native height of the current content |
| long | ContentBitrate | The bitrate-in-use of the current content |
| string | ContentTitle | The title of the current content |
| string | ContentUrl | The URL that the current content was received from |

**Condition operators:**

|  |  |  |
| --- | --- | --- |
| EQ | Equals | (p == v) |
| NEQ | Not equals | (p != v) |
| GTR | Greater than | (p > v) |
| GEQ | Greater than or equals | (p >= v) |
| LT | Less than | (p < v) |
| LEQ | Less than or equals | (p <= v) |
| MOD | Modulo | (p % v == 0) |

**Sources:**

A trigger must have one or more source. The source provides a Uri (typically a Url) that points to a document describing the ad itself. By default, VAST 2.0 ad sources are supported but you could build your own plugin that handles ads described in a different format. *More information is provided below to learn how to do this.*

Additionally, a trigger may have multiple sources. If multiple sources are specified, the ads will play one after another. For example, if you have two sources pointing to a VAST ad, they will be played sequentially.

Lastly, any individual source may have child sources. A child source implies a dependency on the parent source. If a child source is specified, it will only be played if the parent source is handled. If the parent does succeed, all children will be played simultaneously. **Note:** With VAST sources you do not need this feature since a VAST ad can contain multiple ads with implied dependencies (for example, a single linear ad with multiple companion ads).

**Targets:**

Each source can have zero or more target. Targets are used to help define where an ad creative should play in your application. A target can also have child targets that imply a dependency. If a parent target is not used to place an ad, any part of an ad that targets a child target will be skipped.

Targets are required in order to play companion ads since we cannot assume where companion ads should be displayed in your app. Additionally, you must create containers in your app that these targets map to in order for them to work. See the targeting section below for more information about targets and how to correctly set up your app to use them.

*Do it yourself sequencing*

Don’t want to use MAST? No problem. The API that the MAST plugin uses to trigger ads is available to anyone and is done through the use of AdMarkers. You can roll your own sequencing engine one of two different ways:

* Create your own MarkerProvider plugin by implementing IMarkerProvider and decorating with the ExportMarkerProvider attribute. Please refer to the **Microsoft.SilverlightMediaFramework.Plugins.Advertising.MastMarkerProvider** source code as an example on how to do this.
* Simply call the new SMFPlayer.ScheduleAdTrigger method. Here’s an example on how to initiate a VAST ad immediately:

smfPlayer.ScheduleAdTrigger("http://adserver.com/vastdoc.xml", "vast");

Overloads of ScheduleAdTrigger are also available to indicate start time and duration as well as pass in a class that implements IAdSequencingTrigger. IAdSequencingTrigger is much more powerful than a simple url because it allows you to pass all the same source and targeting information that you can specify in a MAST document. Please refer to the section on MAST for more information about how sources and targets can be expected to be handled.

**Targeting**

While linear ads and nonlinear ads are usually played inside the main video area, companion ads need to be placed in specific regions within your application. Targets are used to help determine where an individual ad should be displayed or if it should be displayed at all. Note: linear and nonlinear ads will always default to the main video area unless a matching target is found.

Each ad source (defined by the IAdSequencingSource interface) optionally includes a list of targets (IAdSequencingTarget) to provide guidance on where to place ads. Each individual target can specify a region and type as well as child targets. Let’s dissect what a target looks like:

**Type (string):** This helps identify which ad should be played inside the target. For VAST ads, the target that is used to play an ad creative is determined by first comparing the ID of the ad creative with this property. If a target is not found, the type of the ad creative is used (this is either ‘linear’, ‘nonlinear’, or ‘companion’). Note: IDs are case sensitive but types are not.

**Region (string):** This is the name of the region or container used to host the ad. Once an ad creative is matched with a target, the region property will be used to look up the actual Silverlight or HTML container by Name (in the case of Silverlight) or ID (in the case of HTML).

**Targets (List<IAdSequencingTarget>):** This is a collection of child targets and implies dependency. Remember, the source can contain information for multiple ads. For example, imagine you had an ad source that pointed to a VAST file containing 1 linear ad and 2 companion ads. You might have a target defined for the linear ad and that target could have 2 children (one for each of the companion ads). If the linear ad fails to be placed in the parent target, the 2 companion ads would also not be placed in the child targets and therefore, the entire ad source would be skipped.

**Pre-defined ad containers:** SMF includes the following ad containers by default…

|  |  |
| --- | --- |
| VideoArea | Targets the area over the video playback area. |
| LowerVideoArea | Targets an area at the bottom of the video playback area. |
| UpperVideoArea | Targets an area at the top of the video playback area. |
| LeftVideoArea | Targets an area at the left of the video playback area. |
| RightVideoArea | Targets an area at the right of the video playback area. |
|  |  |

Additionally, you can create your own ad containers by simply adding controls to the SMFPlayer.Containers collection. The name of the control is used as its ID so be sure to always name every control you add. The following FrameworkElements are supported as ad containers: Any panel (e.g. Grid and StackPanel), Border, or ContentControl.

Here’s a practical example of how to control where a companion ad will be seen using MAST & VAST:

*In your MAST document…*

<source uri="vast2-example.xml" format="vast">

  <sources />

  <targets>

    <target region="TopBanner" type="banner" />

  </targets>

</source>

*In your application…*

<Grid x:Name="LayoutRoot">

    <Grid.RowDefinitions>

        <RowDefinition Height="120" />

        <RowDefinition Height="\*" />

    </Grid.RowDefinitions>

    <Grid Grid.Row="0" x:Name="TopBanner" />

    <smf:SMFPlayer Grid.Row="1" x:Name="player" />

</Grid>

public MainPage()

{

    InitializeComponent();

    player.Containers.Add(TopBanner);

}

*In your VAST document…*

<CompanionAds>

  <Companion id="banner" width="300" height="120">

    <StaticResource creativeType="image/jpeg"><![CDATA[…]]></StaticResource>

  </Companion>

</CompanionAds>

**Handling ads**

Once your ads are triggered, the next task is to actually determine what ads to play. The Silverlight Media Framework now includes a plugin to do this for VAST 2.0 formatted ads. Simply add the **Microsoft.SilverlightMediaFramework.Plugins.Advertising** assembly to your project references and this plugin will do the rest for you.

Under the hood, when it is time to play an ad, the Silverlight Media Framework looks for a plugin that implements the IAdPayloadHandlerPlugin interface. At the appropriate time, it passes in the IAdSequencingTrigger object that you scheduled (normally this is created and scheduled for you by the MAST plugin), and expects the ad payload handler plugin to do the rest. The VastAdHandler that ships with SMF is capable of playing ad sources that have a format of “vast”. Internally, the VastAdHandler uses VPAID plugins to actually play the ad described in the VAST document.

*VAST*

VAST is an IAB (Interactive Advertising Bureau) standard that defines an XML-based ad response and schema for in-stream video players. This tells us what ad to play as well as how to track that ad. Please refer to the [IAB website](http://www.iab.net/vast) for more information about the VAST specifications. VAST 2.0.1 is the version currently supported by the Silverlight Media Framework and is the latest version of VAST at the time this document was written.

*VAST documents*

VAST documents are XML files that adhere to the [VAST schema](http://www.iab.net/media/file/VAST_2_0_1_Schema.zip). Most modern ad services use VAST documents to describe video ads. Let’s dissect some key pieces of a VAST document:

Multiple ads. A VAST document can contain multiple ads. These ads are played sequentially in what we term an ad pod.

Each ad can have multiple creatives. A creative can either be linear, nonlinear (overlay), or a companion ad.

When a linear ad is played, your current video is paused and the ad plays in its place. This is much like a traditional commercial on your TV. However, a linear ad isn’t always just a video and can be any type of ad such as an interactive ad with a dynamic duration.

A single linear ad can be described with many different media formats. Often, there are different videos available in different formats to ensure that the player trying to play the ad has choices for which ad to play. SMF has a custom algorithm that it uses to choose the most appropriate ad based on the current size of the video area, the current bitrate of your video, and the type of media. The goal is to ensure that the user always sees the ad of the highest fidelity for their circumstances.

Nonlinear ads are played at the same time as your main video. These are usually positioned on top of your video and off to the side, top or bottom of the video area. Nonlinear ads can also be videos themselves, images or interactive ads.

Companion ads are extra ads that are associated with a linear or nonlinear ad. For example, you might see a banner at the top of an application while a video advertisement is playing at the same time.

Tracking information is also supplied in the VAST document along with each creative. These are simple URLs that will be hit when certain events occur. There are a dozen or so different events that can be tracked such as clicking on the ad, starting the ad, when the ad is half finished, …etc. Please refer to the [VAST specs](http://www.iab.net/media/file/VAST-2_0-FINAL.pdf) for more information.

VAST documents also support the concept of wrapper ads. These are placeholders for other VAST documents. This is useful because it allows an ad server to point to a secondary ad server (usually owned by an individual advertiser) that will actually serve the VAST document. This is much like a redirect except it allows tracking rules to be defined by the primary and secondary ad server. Technically, a secondary ad server could point to a third ad server by returning its own wrapper ad and so on.

**Playing ads**

Once the VAST ad handler is triggered, loads and parses the VAST document, and selects which ad creative(s) to play, it needs an ad player to actually play that ad. This is done through the use of VPAID plugins. The Silverlight Media Framework ships with VPAID plugins capable of playing progressive and adaptive videos, images, HTML, and Silverlight ads.

*VPAID*

VPAID is an IAB standard like VAST but does not define an XML schema. Instead it defines an interface that all ad players must implement in order to play ads. Refer to the [IAB website](http://www.iab.net/vpaid) for more information about VPAID.

The Silverlight Media Framework supports VPAID 1.1 (which is the latest version of VPAID at the time this document was written). VPAID is used by the VAST ad handler to play all ad types (linear, nonlinear, and companion ads). SMF ships with VPAID plugins capable of playing progressive and adaptive videos, images, HTML, and Silverlight ads but you can easily create your own plugins to implement VPAID plugins of other types.

The idea behind VPAID is that it defines a number of properties and methods for the application to communicate to the ad player such as init, start, stop, resize and to set the volume. Additionally, VPAID defines a number of events that it can fire to tell the application that the ad started, stopped, what its progress is or that it was clicked, …etc.

*Extra VPAID plugins*

**Smooth streaming ads:** By default, the advertising plugin uses the active media plugin to play linear ads that target ‘VideoArea’. *Note: ‘VideoArea’ is also the default target for linear ads that don’t explicitly specify a target.* Depending on which media plugin you are currently using to play your main video, you may or may not be able to play ads inside the active media plugin. For example: the assembly **Microsoft.SilverlightMediaFramework.Plugins.Progressive** does not support inline ads. Fear not! There is another way. If the active media plugin does not support ads, another VPAID plugin is available by default to play progressive videos.

Also, a separate plugin: **Microsoft.SilverlightMediaFramework.Plugins.Advertising.Adaptive** is available to play smooth streaming ads. All you have to do is include this assembly in your project and it will be used whenever the ActiveMediaPlugin cannot. Note: These extra VPAID plugins are also used to play companion and nonlinear video ads.

**Silverlight ads:** Many ads today are interactive and act as little tiny separate applications. Many ads use Flash to do this, however, there are various performance considerations you take when combining Flash with Silverlight. A better way to serve interactive ads in your Silverlight application is through the use of native Silverlight ads. SMF provides a way to support this by shipping with an optional plugin called Microsoft.SilverlightMediaFramework.Plugins.Advertising.Xap. This plugin is capable of receiving the URL for a .xap file, downloading the xap file dynamically and playing it as an ad.

**How to build a Silverlight ad:** The process is fairly straightforward.

* Create a new Silverlight application.
* Copy IVPaid.cs into your project. Note: You can find this file in the source code for the project: **Microsoft.SilverlightMediaFramework.Plugins.Advertising.Xap**
* *Optionally, instead you could reference the Microsoft.SilverlightMediaFramework.Plugins.Advertising assembly and use the compiled IVPaid interface instead. However, this is not recommended if you wish to make sure your Silverlight ads will be compatible with other, non SMF player based apps.*
* Add a reference to Microsoft.ComponentModel.Composition (ships with the Silverlight SDK).
* After adding the reference, select it in Visual Studio and set Copy Local = false. This simply helps reduce the size of your .xap ad.
* Create a new UserControl that implements the IVPaid interface.
* Follow the [VAST guidelines](http://www.iab.net/vpaid) from the IAB website to learn about what each property, method and event can be used for.
* Decorate your UserControl with the following attributes:

    [Export("IVpaid")]

    [PartCreationPolicy(CreationPolicy.NonShared)]

    public partial class MainPage : UserControl, IVpaid

* Go to your project properties and change the Startup object to ‘(not set)’
* Delete App.xaml from your project.

Compile your new ad and put it on your server. Be sure to add a clientaccesspolicy.xml file to your server’s root so apps from other domains can retrieve the ad. This ad is now capable of being consumed as an ad and will be given priority over all other media types in SMF.

**Security Warning:** If you add Microsoft.SilverlightMediaFramework.Plugins.Advertising.Xap to your project, you are enabling the ability to play Silverlight ads inside your app. These ads will run inside your application just as your own UserControls would, so if you do not trust the source of the ads AND your app has access to sensitive information, it is technically possible for the ad to do something malicious to your app.

Examples of things the ad could do:

* Access any public member in your app.
* Call your server using the current user’s security credentials.
* Alter the visual tree of your application in some way.
* Note: this is not meant to be a complete list. It is recommended you perform your own security analysis before using.

However, to put this into perspective: VAST also supports Javascript ads to be run in your html page so you may find that allowing Silverlight ads does not actually make your app any more vulnerable.

**Advertising Definitions**

**Companion Ad:** Commonly text, display ads, rich media, or skins that wrap around the video experience. These ads come in a number of sizes and shapes and typically run alongside or surrounding the video player.

**InLine Ad:** VAST document that includes all the elements necessary to display the visual experience of the ad.

**Linear Video Ad:** The ad is presented before, in the middle of, or after the video content is consumed by the user, in very much the same way a TV commercial can play before, during or after the chosen program.

**Non-linear Video Ad:** The ad runs concurrently with the video content so the users see the ad while viewing the content. Non-linear video ads can be delivered as text, graphical ads, or as video overlays.

**Primary Ad Server:** First ad serving system called by the Video Player or other framework. It is assumed that in most cases a publisher will make all initial ad requests through their Primary Ad Server (whether homegrown or third party), then redirect to other ad severs as needed.

**Secondary Ad Server:** Ad server used by an ad network or by the buyer of ads to serve creative, track results and optimize creatives.

**VAST (Video Ad Serving Template):** XML document format describing an ad to be displayed in, over, or around a Video Player or a Wrapper pointing to a downstream VAST document to be requested.

**Video Player:** Environment in which in-stream video content is played. The Video Player may be built by the publisher or provided by a vendor.

**Wrapper Ad:** VAST document that points to another VAST document from a different server.