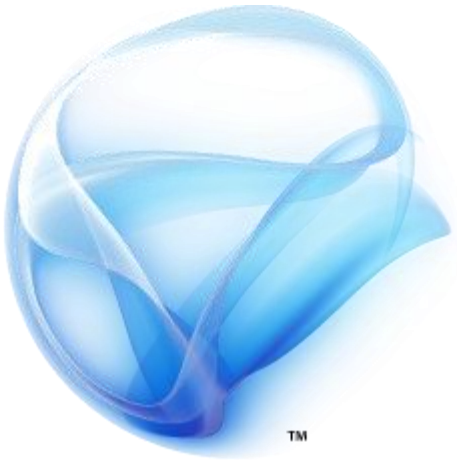
Silverlight Ad Publishing & Serving Guide



Microsoft® Silverlight™

Contents

Overview 4

What is a XAP? 5

Silverlight Versions 6

Web Server Changes – Configuring MIME type 7

Enabling Cross-Domain Access 8

- Flash Cross-Domain compatibility 8
- How ClientAccessPolicy.xml works 9
  - ClientAccessPolicy.xml Example: Unrestricted Access 9
  - ClientAccessPolicy.xml Example: Restricted Access 10
- Cross-Domain Requirements for Publishers 10
- Additional Cross-Domain Items of Interest 11

Ingesting Silverlight Ads into a Creative Management System 12

- Enabling XAP File Format 12
- Validating the Creative 12
- Handling “Deep Zoom” Ads 13

Configuring the Silverlight OBJECT tag 14

Analytics Tracking 16

Ad Synchronization 17

- Creating a JavaScript function within the HTML 18
- Calling a JavaScript function from within an ad 18
- Calling a function within an ad from JavaScript 18
- Additional Ad Synchronization Information 19

Passing Startup Parameters to an Advertisement 20

- Using initParams to Pass Information 20
- Load Settings from an XML File 21

Silverlight Installation Detection and Ad Degradation 22

- Detecting Browser Capabilities 22
- Adding Experience Degradation 23
  - Silverlight to Flash to IMG 23
  - Silverlight to IMG 23

Overview

This document provides guidance to advertising networks, analytics providers, and publishers for publishing and serving Silverlight-based advertisements. It includes most of the information you need to ingest an ad into a creative management system, serve the ad, publish it on a web page and track analytics against it. This document is intended for the following audiences:

- Those intending to ingest Silverlight ads into an ad serving creative management system.
- Those intending to serve Silverlight ads from an ad server/network.
- Those intending to publish Silverlight ads along with other content on a web page.
- Those intending to accept Silverlight ad tracking or media tracking information.

This document is intended to supplement any SDKs or tools provided by publishers or ad networks that may be specific to their implementation.

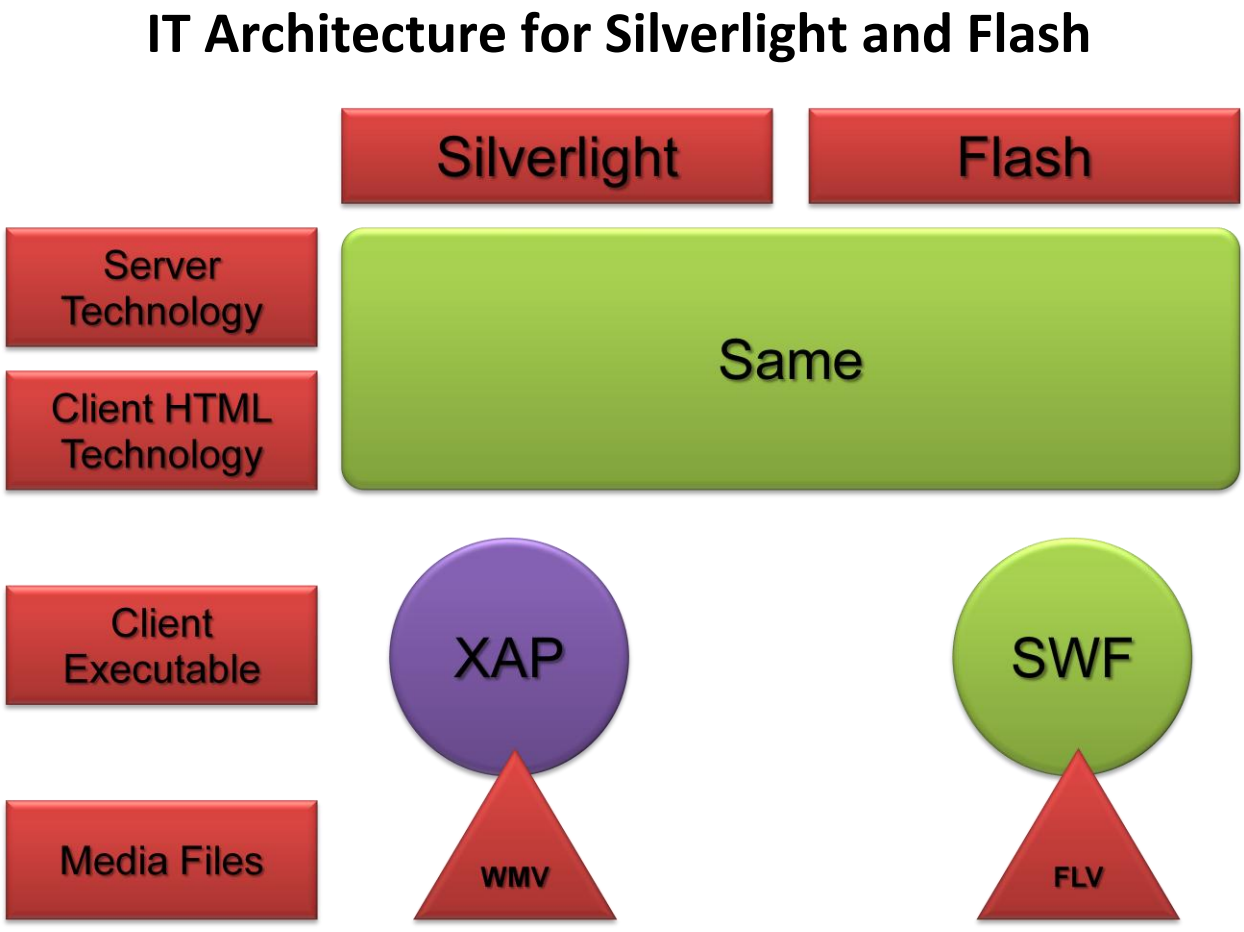
Note: The features and concepts mentioned here are a subset of the Silverlight set of features. Therefore, more information can be found by consulting other documentation specific to an area.

What is a XAP?

A XAP (pronounced “ZAP”) is the file format for Silverlight 2 applications. A XAP file is actually just a ZIP file that contains the application and all the resources it needs. To prove this concept, change a XAP file extension to ZIP and then open it up. What you will see are several files required to run the application.

Since people are more familiar with Adobe Flash and how it functions within their infrastructure, it’s important to understand what the differences between Silverlight and Flash are from an IT perspective. Despite the two technologies being quite different, they are similar from a runtime perspective and Silverlight should be able to integrate fairly easily into an existing Flash environment.

As the following diagram shows, both Silverlight and Flash utilize the same web server technology and are hosted within a web page in a similar manner. The differences come in the file formats. A Flash SWF file is synonymous with a Silverlight XAP file. A Flash video file (FLV) is a container for video, similar to a Windows Media file (WMV/ASF/ASX/WMA/etc.)



Silverlight Versions

Silverlight 1 (released in September 2007) is a browser plug-in that renders rich applications with a JavaScript-based programming model. Silverlight 2 was released in October of 2008 and includes a number of changes that make delivering ads more feasible.

- Support for a subset of the .NET Framework
- Programmable with managed languages (C#, Visual Basic, etc.)
- Packaging applications into XAP files
- Compelling new features, such as Deep Zoom image handling

While Silverlight 2 is backward compatible to version 1, ads should only be deployed using the latest version.

When ingesting ads into a creative management system, it is recommended that some validation is performed to ensure that the ad was created with the proper version. To verify the version number used to build the application, perform the following steps.

IMPORTANT: Silverlight 2 beta versions are not forward compatible with the released version and will therefore not function. Beta versions include any version number below "2.0.31005.0".

1. Change the XAP file extension to ZIP. For example, MyAd.xap should be changed to MyAd.zip.
2. Open the ZIP file and view the contents of AppManifest.xaml.
3. Verify that the **RuntimeVersion** setting is set to at least **"2.0.31005.0"** as the following highlighted example shows.

```
<Deployment
        xmlns="http://schemas.microsoft.com/client/2007/deployment"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        EntryPointAssembly="MyAd"
        EntryPointType="MyAd.App"
        ExternalCallersFromCrossDomain="FullAccess"
        RuntimeVersion="2.0.31005.0">
        <Deployment.Parts>
                <AssemblyPart x:Name="MyAd" Source="MyAd.dll" />
        </Deployment.Parts>
</Deployment>
```

Web Server Changes – Configuring MIME type

Silverlight is compatible with all popular web servers; however, the web server must be configured to allow Silverlight files to be downloaded. This is a fairly straight-forward process of adding MIME types, but the process is different for each web server. Therefore, you should consult your web server documentation for details about how to add MIME types.

Add the following MIME type:

File Extension	MIME Type Setting
XAP	application/x-silverlight-app

In addition to the above setting, there may be scenarios where loose XAML files may need to be downloaded directly. Examples for this may be a customized splash screen or localized settings. Therefore, it is recommended to also add the following **optional** MIME type:

File Extension	MIME Type Setting
XAML	application/xaml+xml

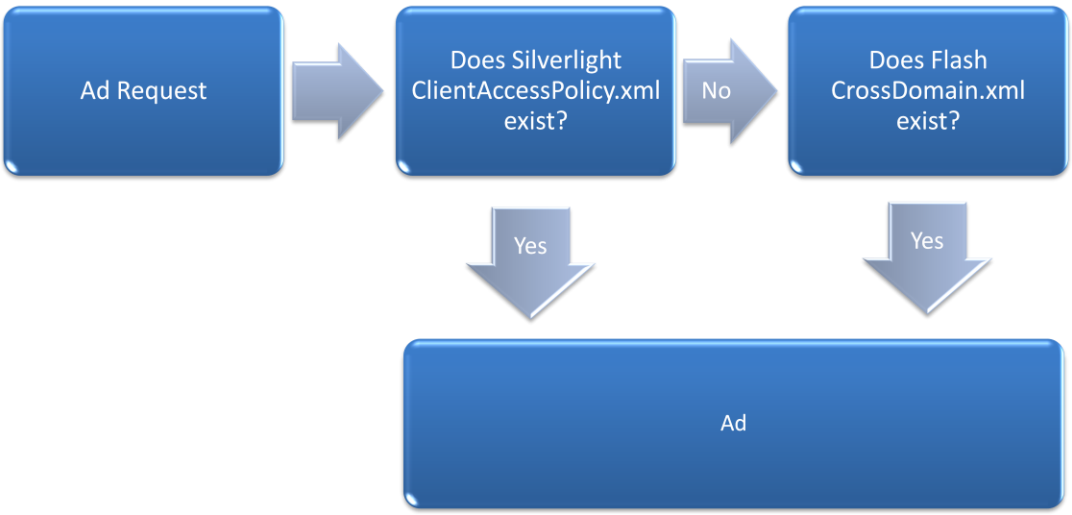
Enabling Cross-Domain Access

A typical web page is hosted on one domain, its ads on another domain and those ads are generally tracked against yet another domain. By default, the Silverlight security model prevents loading of data from multiple domains. Adding a cross-domain policy file on the domain that you need to access allows you to make exceptions to this security model. This section discusses how to configure the server to allow ads to be tracked and hosted cross-domain.

Flash Cross-Domain compatibility

Flash ad providers have had to deal with cross-domain access for years and as a result most ad domains use a Flash policy file. If a basic Flash policy file exists (CrossDomain.xml), Silverlight will honor it. Therefore, it is quite likely that nothing will need to be done to support Silverlight ads. However, there are two important considerations/issues:

1. **Compatibility** – Silverlight only supports basic Flash CrossDomain.xml files that mark an entire domain as public. Any other scenarios (allowing access to partner domains, allowing access to a sub-path, enabling certain ports, allowing sockets, etc.) are not supported using the Adobe format.
2. **Performance** – When a web request is made, Silverlight first looks for a ClientAccessPolicy.xml file. If it finds the file, it honors its settings and does NOT check for Flash's CrossDomain.xml. However, if ClientAccessPolicy.xml is NOT found, a **second** request is made to the server for CrossDomain.xml. The initial check for the Silverlight-only version of the file adds a few milliseconds (or possibly more depending on network bandwidth) to the initial request. Once the cross-domain check is completed, the results of the check is cached on the client for the remainder of the time the application is running. Once the application is closed (page is unloaded from the DOM, a different web page is displayed, the browser is closed, etc.) the settings are lost and will be requested again the next time the application is loaded. For these reasons, it is recommended that a Silverlight policy file be utilized. The following diagram depicts the policy file checking behavior.



How ClientAccessPolicy.xml works

The policy file should be placed in the root of the domain and named ClientAccessPolicy.xml. Silverlight only searches the domain root so if this file is in another location, it won't be found.

It is a good idea to turn off client caching of the policy file by configuring the web server (IIS, Apache, etc.) to set a **Cache-Control:no-cache** response header on the policy file. Consult your web server documentation for details about how to accomplish this. Otherwise changes made to the file over time may not be read.

ClientAccessPolicy.xml Example: Unrestricted Access

The following example allows unrestricted access to the domain and all subfolders within it.

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
        <cross-domain-access>
<policy>
        <allow-from>
                <domain uri="*"/>
        </allow-from>
        <grant-to>
                <resource path="/" include-subpaths="true"/>
        </grant-to>
</policy>
                </cross-domain-access>
</access-policy>
```

ClientAccessPolicy.xml Example: Restricted Access

The following example limits access to specific directories on the server.

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
        <cross-domain-access>
<policy>
        <allow-from>
                <domain uri="http://MyPrivateDomain.com"/>
        </allow-from>
        <grant-to>
                <resource path="/DisplayAds" include-subpaths="true"/>
                <resource path="/InstreamAds" />
        </grant-to>
</policy>
                </cross-domain-access>
</access-policy>
```

Cross-Domain Requirements for Publishers

Running ads (XAPs) from another domain requires that the **source** parameter to be set to a fully qualified URI as the following example shows:

```
<object data="data:application/x-silverlight,"
      type="application/x-silverlight-2" width="100%" height="100%">
      <param name="source" value="http://AdServer/MyAd.xap"/>
      <param name="enableHtmlAccess" value="true" />
</object>
```

While not specific to cross-domain concerns, the **enableHtmlAccess** parameter should also be specified so that Click-Thrus can occur. Additionally, if the ad must access JavaScript functions on the page (such as analytic tracking libraries), this option must be specified. The following shows an example:

```
<object data="data:application/x-silverlight,"
      type="application/x-silverlight-2" width="100%" height="100%">
      <param name="source" value="http://AdServer/MyAd.xap"/>
      <param name="enableHtmlAccess" value="true" />
</object>
```

If any JavaScript on the web page will be calling functions within a cross-domain creative, the creative must be configured to allow for this. It is a good idea for the ad network or publishers ingestion workflow to validate this setting. To verify this setting, do the following:

1. Change the XAP file extension to ZIP. For example, MyAd.xap should be changed to MyAd.zip.
2. Open the ZIP file and view the contents of AppManifest.xaml.
3. Verify that the **ExternalCallersFromCrossDomain** setting exists and is set to **"FullAccess"** as the following highlighted example shows.

```
<Deployment
        xmlns="http://schemas.microsoft.com/client/2007/deployment"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        EntryPointAssembly="MyAd"
        EntryPointType="MyAd.App"
        ExternalCallersFromCrossDomain="FullAccess"
        RuntimeVersion="2.0.31005.0">
        <Deployment.Parts>
                <AssemblyPart x:Name="MyAd" Source="MyAd.dll" />
        </Deployment.Parts>
</Deployment>
```

Additional Cross-Domain Items of Interest

Given the information above, you might think that the cross-domain file is checked each time a request is made to a server resulting in a potential performance issue. However, this is not the case. The policy file is read only ONCE per application (ad) instantiation. All subsequent server calls use the result from the initial request. However, if the application is closed and then reloaded, it will request the policy file again.

There are several other features of the policy file. For more information, see the "Networking and Communication" section in the Silverlight documentation (http://msdn.microsoft.com/en-us/library/cc645029(VS.95).aspx).

Ingesting Silverlight Ads into a Creative Management System

Since many of the files in Silverlight parallel Flash, very little additional work must be done to enable the ingestion of Silverlight ads into a creative asset management system. However, the following are three differences:

1. Enabling XAP file format
2. Validating the creative
3. Handling “Deep Zoom” ads

> Deep Zoom is a technology that provides the ability to zoom large images in an efficient manner. Normally, loading large images is not an optimal experience for users because of the time required for the images to load. Deep Zoom mitigates this difficulty by loading progressively higher resolution images. This creates a "blurry to sharp" experience for users. The progressive image files are stored in a folder on the server. Since ingesting this potentially large number of files is not the norm today, it is being called-out in this documentation as an area to be aware of.

Let’s look at each of the items in more detail.

Enabling XAP File Format

Enabling the XAP file format is perhaps the easiest change to account for. Typical ingestion systems are web based and contain a web-based form for specifying the file(s) to upload. Instead of specifying a SWF or FLV for Flash, the uploading of XAP and WMV files would be allowed.

Validating the Creative

Validating a set of requirements set forth by the ad network and publisher is a common step when ingesting ads. Some of the validation may be manual; however, programmatic validation provides the most efficiency and effectiveness for a majority of typical requirements.

Requirements for the creative typically consist of the following:

- File size. File sizes may need to be within allowed boundaries.
- Dimensions. Dimensions of ads may need to conform to allowed height and width. The dimensions of the ad can be specified on the object tag, however, that does not mean the ad is that size. It just means that the ad will appear within a window of that size and will be visually clipped if the sizes vary.
- Frame rates. Frame rates may be limited to a certain rate (15 FPS for example).
- Version. Ads may need to be built with a certain software version.
- Media bit rate, duration and resolution.
- Volume levels.
- Background color.
- Initial animation duration.

- Use of certain methods, properties, etc.

Validating several of these items is fairly straightforward (using the file system to check ad size for example), while others require access deep into the application. Vivek Dalvi, Principal Group Program Manager for Microsoft's Silverlight Development team has a blog where he discusses how to dissect a Silverlight application to gain access to the individual data points for validation. For example, a list of all APIs used in the ad as well as access to all media and markup (XAML) can be obtained. Vivek's blog can be found at http://vivekdalvi.wordpress.com.

Handling "Deep Zoom" Ads

Deep Zoom can display large images in an efficient manner and subsequently allow you to zoom into an image. Deep Zoom has sparked great interest by advertisers who wish to show detailed images of their products (for example, the fine stitching detail on a leather seat in a car).

Part of the process for making this remarkable feature work involves breaking up larger images into smaller pieces that are loaded as-needed from the web server. The number of files in a "Deep Zoom" ad can vary significantly, but suffice it to say that an ad could have hundreds of image files.

Obviously, entering each filename in a web form for ingestion is undesirable, so the following steps are recommended to make the process easier.

1. Locate the folder containing the Deep Zoom files. All Deep Zoom files are typically located in a folder named GeneratedImages that resides in the same location as the XAP.
2. Compress the GeneratedImages folder using zip-compression or any other agreed-upon compression format.
3. Submit the compressed file along with the accompanying XAP.
4. Decompress the file and place the results in the proper location on the ad server.

Configuring the Silverlight OBJECT tag

While there is more than one supported way to instantiate a Silverlight 2 application, the **object** tag method is the easiest and most reliable for publishers.

The following example shows the minimum requirements for a 728x90 banner ad.

```
<object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2" width="728" height="90">
      <param name="minRuntimeVersion" value="2.0.31005.99"/>
      <param name="autoUpgrade" value="false"/>
      <param name="source" value="MyAd.xap"/>
</object>
```

If the web page should trap any unhandled errors that occur within the Silverlight application, the **onerror** parameter can be set to a value that points to a JavaScript function to call.

```
<object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2" width="728" height="90">
      <param name="source" value="MyAd.xap"/>
      <param name="minRuntimeVersion" value="2.0.31005.99"/>
      <param name="autoUpgrade" value="false"/>
      <param name="onerror" value="onSilverlightError" />
      <param name="enableHtmlAccess" value="true" />
</object>
```

If the application must communicate with the HTML DOM then **enableHtmlAccess** must be set to **true**, otherwise it can be specifically disabled by setting it to **false**.

It is important to note that by default, if the XAP is loaded from the same domain as the web page then DOM access is enabled. However, if the XAP is loaded from another domain (cross-domain) then DOM access is disabled. Specifying this parameter overrides the default behavior.

Another important point to understand is that Silverlight utilizes the DOM for handling Click-Thrus. If DOM access is disabled, Click-Thrus will NOT occur. Therefore, it is recommended that **enableHtmlAccess** always be set to **true**.

```
<object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2" width="728" height="90">
      <param name="source" value="MyAd.xap"/>
      <param name="onerror" value="onSilverlightError" />
      <param name="minRuntimeVersion" value="2.0.31005.99"/>
      <param name="autoUpgrade" value="false"/>
<param name="enableHtmlAccess" value="true" />
</object>
```

Since publishers should only attempt to instantiate Silverlight applications if the Silverlight runtime is installed, there is no need to provide links to install Silverlight as is commonly the case with other usage scenarios.

Analytics Tracking

Silverlight has a very rich event model as well as the ability to expose a plethora of information about how the user is engaging with the application. As advertisers continue to drive for an increased ROI, the ability to track, analyze, and report on activities becomes more important.

There are a few things to keep in mind when tracking information from advertisements.

Cross-Domain Enablement – If data is being tracked via the same domain as the creative was served, then no cross-domain policies are checked. However, if data is being tracked by a secondary provider (i.e., a third-party analytics provider, such as WebTrends or Omniture) then cross-domain policies are in-effect and a cross-domain policy file must be placed on the tracking server. The following table describes aspects of what is allowed for cross-domain requests.

Category	Description
Allowed File Schemes	HTTP, HTTPS & FILE
Cross-Scheme	This is not supported. If the web page is HTTPS it is not possible to access an image on an HTTP server and vice-versa.
Cross-Domain	No HTTPS. Only HTTP is supported cross-domain.
Redirection	Yes. Redirection is allowed for HTTP but not for HTTPS.

Tracking Methods – Tracking events (impressions, for example) are done in a variety of ways. A few of the more popular methods are fully supported by Silverlight. The following is a list of these methods along with some points that should be taken into consideration:

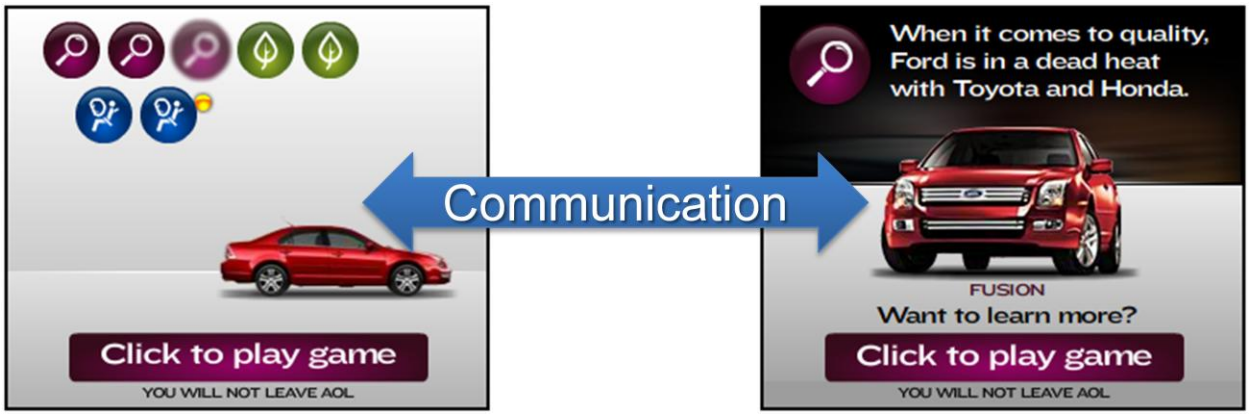
- Calling a JavaScript function that sets the source property of a 1x1 transparent GIF.
    - Requires that the creative be granted access to the DOM.
    - Very popular method among publishers and third-party analytic providers.
    - Allows the publisher to "police" all tracking through one central location.
    - GIF calls are handled efficiently on the web server.
    - Tracking data length is limited to URL limits (~2048 characters).
- Setting the source property of a 1x1 transparent GIF from within the creative.
    - Removes the need for the publisher to "police" tracking.
    - GIF calls are handled efficiently on the web server.
    - Tracking data length is limited to URL limits (~2048 characters).
    - Cross-domain enablement is **NOT** required.
- Calling a web method to post data.
    - Removes the need for the publisher to "police" tracking.
    - Posting data is not as fast as handling GIFs.
    - Tracking data length is unlimited.
    - Cross-domain enablement is required.

Ad Synchronization

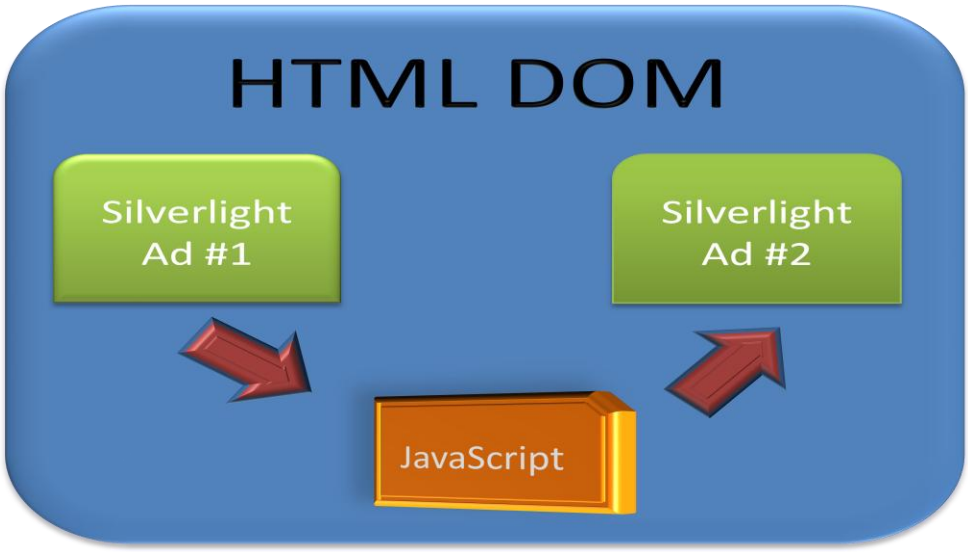
Ad synchronization allows multiple ads running on a single web page to communicate with each other. This is useful if you have multiple creatives with a common theme, such as a car driving across the screen from one ad into the other. Another common example for in-stream ads is to display a companion ad (a banner for example) while a related ad is played within the media player.

While most of the work involved in synchronizing ads will be done by the designer/developer, it is important for the publisher to understand how the HTML DOM interaction works so that they can provide guidance to the designers.

Flash designers/developers may be familiar with a Flash feature called LocalConnection which does essentially the same thing.



Ads can be synchronized in Silverlight by sending a message from one ad to the HTML DOM and then having that message relayed to the other ad using JavaScript. The following graphic shows the flow of information from one ad to a JavaScript function and then into another ad.



The following sections demonstrate one method for enabling two ads to communicate by using JavaScript. These sections show only a very basic communication mechanism. A more robust mechanism is listed in another document titled "Silverlight Advertising Creation Kit", which is available from the same web site that this document was obtained. For more information about calling JavaScript from managed code and calling managed code from JavaScript, see the HTML Bridge topics in the Silverlight documentation (http://msdn.microsoft.com/en-us/library/cc645076(VS.95).aspx).

Creating a JavaScript function within the HTML

The following JavaScript function named *myFunction* in the HTML can be called by the creative.

```
<script type="text/javascript">
      function myFunction (   parameter1,
                              parameter2,
                              parameterEtc
                            )
      {
            // Insert code to pass info to the other ad
      }
</script>
```

Calling a JavaScript function from within an ad

The following C# code shows how to call a JavaScript function named *myFunction* from within an ad:

```
using System.Windows.Browser; // Required for access to DOM
namespace MyCoolAd
{
    public class MyAd
    {
       private void CallJavaScriptFunction()
       {
            HtmlPage.Window.CreateInstance("myFunction",
                                           parameter1,
                                           parameter2,
                                           parameterEtc
                                          );
       }
    }
}
```

Calling a function within an ad from JavaScript

The following shows a C# function named *DoSomething* that resides within the ad. This code has been configured so that it can be called from JavaScript.

```
namespace MyCoolAd
```

```
{
    [ScriptableType]
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
            HtmlPage.RegisterScriptableObject("SilverlightAd", this);
        }

        [ScriptableMember]
        public void DoSomething()
        {
            // Put code here to handle request from DOM
        }

}
```

To call the *DoSomething* function from JavaScript, you need to specify the Silverlight **object** tag name (see the following highlighted reference). In this example, the **object** tag name is *silverlightObject*. Additionally, you need to specify the name of the Silverlight application that is being called into. In this example the name *SilverlightApp* was used. Finally, the *DoSomething* function name is specified.

```
<object id="silverlightObject" data="data:application/x-silverlight,"
      type="application/x-silverlight-2-b2" width="400" height="300">
</object>

<script type="text/javascript">
      silverlightObject.Content.SilverlightApp.DoSomething();
</script>
```

Additional Ad Synchronization Information

If an ad is hosted in an **iframe**, there are browser security restrictions to contend with. For example, an ad hosted within an **iframe** cannot access its parent (and therefore cannot access another ad); however, the parent can access its children within the **iframe**. In other words, a Silverlight application located in the main HTML DOM can access other Silverlight applications contained within **iframes** but the main Silverlight application cannot.

Passing Startup Parameters to an Advertisement

Depending on the implementation, it may be necessary to pass certain information from the ad server or publisher into the application. This might be necessary for specifying a clickTAG, an ad ID number, tracking server, etc.

There are two recommended approaches:

1. Use the **initParams object** tag parameter to pass information during application load. Detailed instructions on how to accomplish this are outlined later in this section.
2. Load settings from an XML (or other file format). This option is more complex; however, it provides a greater degree of data and doesn't clutter the object tag.

Both of these approaches require that the creative contain code for retrieving the information being passed to it. Therefore, the publisher and/or ad network should communicate to the creative agency which approach is being used and what to do with the information being passed. It is also possible to use a combination of approaches rather than just rely on one.

Using initParams to Pass Information

Like Flash, Silverlight has the ability to read parameters from the object tag used to instantiate the Silverlight ad. Silverlight supports a parameter named **initParams** which can include any number of name/value pairs. Using **initParams** is fairly straight-forward and is the preferred method to use if dynamic data must be passed to the application. One of the more common requirements is the specification of a *clickTAG*. Details for this scenario are discussed more in this section; however, the principles apply to any information you wish to pass in this manner.

A *clickTAG* is a unique tracking identifier assigned by an ad serving network to an ad. The *clickTAG* allows the ad network to track where the ad was displayed when a user clicks on it by sending the *clickTAG* information to a tracking server.

The following example shows how to include the *clickTAG* parameter. Note also that HTML access needs to be granted to the ad so that navigation to the *clickTAG*'s specified URL can occur.

```
<object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2 " width="100%" height="100%">
      <param name="source" value="MyAd.xap"/>
      <param name="onerror" value="onSilverlightError" />
      <param name="minRuntimeVersion" value="2.0.31005.99"/>
      <param name="autoUpgrade" value="false"/>
      <param name="enableHtmlAccess" value="true" />
      <param name="initParams" value="clickTAG=http://MyAdTracking
            TrackingServer/track?http://NewDestination.htm" />
</object>
```

In addition to specifying the information above, the creative must contain code to specifically read the parameter information and process it when necessary. Therefore, guidance should be provided to the designer/developer as to the proper information to parse.

The **initParams** parameter can specify several different name/value pairs so it's possible to send additional data to the ad during object instantiation. To specify additional parameters, use a comma to delimit the name/value pairs as the following example shows:

```
<object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2 " width="100%" height="100%">
      <param name="source" value="MyAd.xap"/>
      <param name="onerror" value="onSilverlightError" />
      <param name="minRuntimeVersion" value="2.0.31005.99"/>
      <param name="autoUpgrade" value="false"/>
      <param name="enableHtmlAccess" value="true" />
      <param name="initParams" value="clickTAG=http://MyAdTracking
            TrackingServer/track?http://NewDestination.htm,
            myParam1=myValue1,myParam2=myValue2"
 />
</object>
```

Load Settings from an XML File

Silverlight provides a way for the publisher or ad network to embed data into the XAP file. Since the XAP is merely a renamed ZIP file, additional files can be easily added to it. Since this process involves adding a file to the XAP, it is not recommended that this procedure be used for dynamic data. It works well for information that is static. For example, you may want to specify several different URLs to use for tracking various events that are specific to an advertisement.

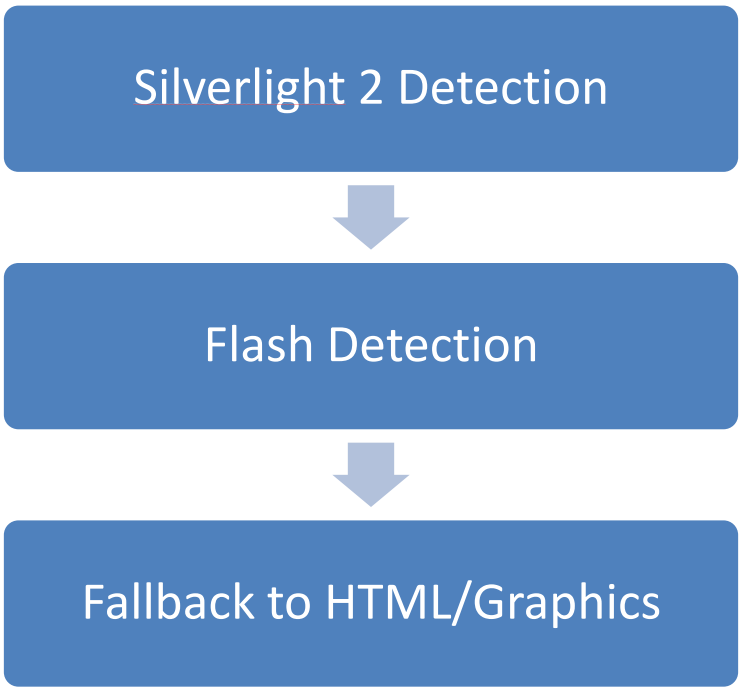
Once the file has been added to the XAP, the creative designer/developer must load the file on startup and parse it. For more information, see the "XML Data" section in the Silverlight documentation (http://msdn.microsoft.com/en-us/library/cc188996(VS.95).aspx).

Silverlight Installation Detection and Ad Degradation

In order for a Silverlight ad to be displayed, Silverlight 2 must be installed on the computer. If it is not installed then the ad format used must be degraded gracefully to Flash or an HTML-style ad. Prompting a user to install Silverlight in order to view an ad is not a desirable user-experience, so it is important that proper degradation steps be adopted.

In most cases, the same ad will not be built for both Silverlight and Flash. Therefore, degradation from Silverlight to Flash will likely be rare and is only demonstrated for completeness.

As the following graphic shows, the page should attempt to load the Silverlight 2 ad first. If Silverlight is not available, then the page can optionally fall back to Flash. If neither rich media formats are available, then HTML-style ads should be used.



Detecting Browser Capabilities

Detecting browser capabilites on the client side is accomplished via cascading **<object>** tags. A high-level example of the HTML looks like the following:

```
<object type="application/x-silverlight-2">
      <object type="application/x-shockwave-flash">
            <img src="ad.gif"/>
      </object>
</object>
```

The browser determines if Silverlight is installed by checking if it can create an object of type **"application/x-silverlight-2"**. If it cannot, it will try to render the inner HTML of that object (ignoring

any **<param>** tags along the way). The next object in line will be of type **"application/x-shockwave-flash"**. If that one is also unknown to the browser, then it will fall back to the **<img>** tag.

Note that some browsers, including Firefox, will download the resources associated with the inner HTML of **<object>** tags. Therefore, you may see server requests for ad.gif even though the image is not being shown to the user.

Adding Experience Degradation

The following sections list examples of how to implement degradation in your HTML page.

Silverlight to Flash to IMG

The following **object** tag shows full degradation from Silverlight to Flash and then finally to an image.

```
<object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2" width="100%" height="100%">
      <param name="source" value="MyAd.xap"/>
      <param name="onerror" value="onSilverlightError" />
      <param name="minRuntimeVersion" value="2.0.31005.99"/>
      <param name="autoUpgrade" value="false"/>
      <param name="enableHtmlAccess" value="true" />
      <param name="initParams" value="clickTAG=http://MyAdTracking
            TrackingServer/track?http://NewDestination.htm,
            myParam1=myValue1,myParam2=myValue2" />
      <object type="application/ x-shockwave-flash" width="100%"
            height="100%">
            <param name="movie" value="MyAd.swf"/>
            <img src="myAd.gif" alt="My Ad" />
      </object>
</object>
```

Silverlight to IMG

The following **object** tag shows degradation from Silverlight to an image.

```
<object data="data:application/x-silverlight-2,"
      type="application/x-silverlight-2" width="100%" height="100%">
      <param name="source" value="MyAd.xap"/>
      <param name="onerror" value="onSilverlightError" />
      <param name="minRuntimeVersion" value="2.0.31005.99"/>
      <param name="autoUpgrade" value="false"/>
      <param name="enableHtmlAccess" value="true" />
      <param name="initParams" value="clickTAG=http://MyAdTracking
            TrackingServer/track?http://NewDestination.htm,
            myParam1=myValue1,myParam2=myValue2" />
      <img src="myAd.gif" alt="My Ad" />
</object>
```