Akamai Media Player - Standard

An Akamai Professional Services Solution

User Guide

Document updated on: July 26, 2011
Document version: v1.02

Player version: 2.0

Table of Contents

OVERVIEW

About the Player

The Akamai Media Player is a feature-rich, Flash-based video player that can be readily deployed onto your web site with minimal configuration or coding.

Features

- Supports playback of single bit-rate (SBR) and multi bit-rate (MBR) streams delivered using the Adobe Flash Media Server Real Time Messaging Protocol (RTMP)
- Supports playback of Akamai HD Network single bit-rate (SBR) and multi bit-rate (MBR) streams – HDN for Flash 1.0 and HDN for Flash 2.0
- Built-in support for Audience Analytics and QoS Monitor
- Built-in support for Akamai Secure Streaming and SWF verification
- Easy player configuration using FlashVars and XML
- Built-in JavaScript API
- Supports DVR functionality for live streams
- Support for Closed Captioning and Accessibility features
- Supports branding and color customizations

USER INTERFACE

Player UI

Fig: 1 – The Player UI

Play/Pause button

This toggle button allows the user to play or pause the video.

Fig: 2 – The Play/Pause button

Rewind button

Clicking this button rewinds the video by an interval specified in the player configuration. The rewind button is enabled for LIVE streams only if DVR is enabled.

Fig: 3 – The Rewind button

Seekbar

The seekbar has 5 main components integrated into it:

1. Progress indicator: Displays the progress of the video. The user is allowed to seek to a point by clicking on the seekbar or by dragging the seekbar thumb to the point. The progress indicator is not available for live streams with DVR not enabled. However, if DVR is enabled, the progress indicator would indicate the position of the live stream.
2. Time indicator: Displays the elapsed time and the total duration of the video

Fig: 4 – The Seekbar with progress indicator and time indicator for Non-DVR streams

.

3. Live Indicator: Displays the text "LIVE" when a live stream is playing.

Fig: 5 – The Seekbar with Live indicator for Non-DVR streams

4. Download progress indicator/bar: This progress bar displays the download progress for a progressive download (PDL) stream.

Fig: 6 – The Seekbar with the download progress indicator for progressive streams

5. DVR seekbar: This seekbar is displayed when the user rewinds or seeks to a particular part of the video. It displays the progress of the video in DVR mode.

Fig: 7 – The Seekbar with for live streams with DVR enabled

6. "Go Live" button: The "Go Live" button appears when a live stream is playing DVR content. On clicking the button, the player will seek to and play live content.

Fig: 8 – The Seekbar with the "Go Live" button for live streams

Volume control

The volume control is a multi-purpose control. When clicked, it toggles between the muted and un-muted states. When rolled over, it displays a slider control for precise manipulation of the player's output volume. The default volume can be set in the configuration file. The theme color of the control is set based on the theme color set in the player configuration.

Fig: 9 – Volume button

Fig: 10 – Volume control slider

Bandwidth button and the Bandwidth Panel

The bandwidth button toggles the visibility of the Bandwidth Panel. The panel displays information about the user's connection speed, the bit rate of the current stream and the maximum bit-rate reached while playing the stream. If the bandwidth-related information is not available, then these fields would be set to "Not available".

Fig: 11 – Bandwidth toggle button

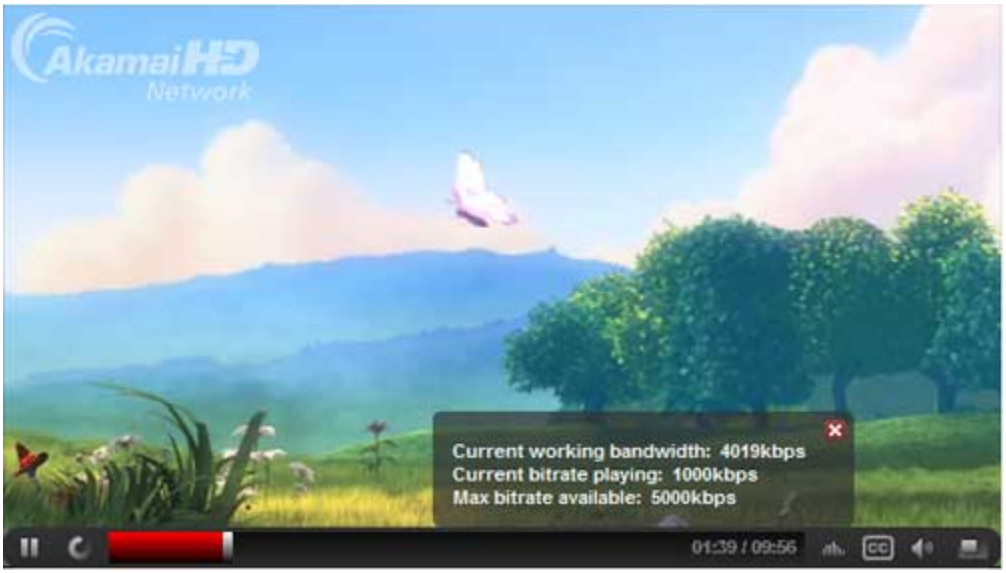


Fig: 12 – Bandwidth meter displaying bandwidth related information

Fullscreen toggle button

This button can be used to toggle the player screen between fullscreen and normal mode.

Fig: 13 – Fullscreen toggle button

Replay button

The replay button is displayed if the player is not in looping mode and the video has reached the end of the stream. The looping mode can be enabled or disabled in the configuration file. For more information, refer to the Configuration section of this guide.

Fig: 14 – Replay button displayed at the end of video stream

Buffering bar

The buffering bar is displayed any time the video player is loading the content to continue playback.

Fig: 15 – Buffering bar

Play Overlay button

The Play Overlay button is displayed on the player UI when the video is paused by the user. The button can be clicked to resume playback.

Fig: 16 – Play Overlay button

Context menu

The player's right click menu contains useful information about the player such as the player version and the versions of plug-ins used.

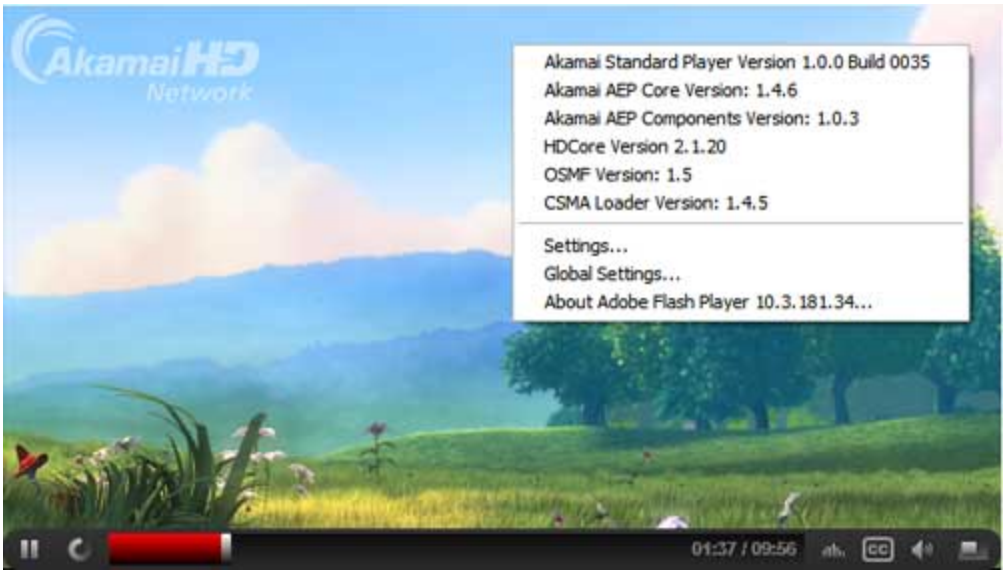


Fig: 17 – Context menu available on right-click

CONFIGURING THE PLAYER

The Akamai Media Player can be configured by setting parameters using flashvars and a configuration XML file. The absolute or relative location of the configuration XML file needs to be specified using the ***settings_url*** parameter in the flashvars.

Flashvars-only Configuration Parameters

The player requires specific parameters during its initialization process namely - ***settings_url, cache_bust_key, location, control*** and ***auto_hide***. These parameters can be specified only using flashvars and should be available to the player when the player loads.

XML-only Configuration Parameters

Certain parts of the configuration such as the ***controls***, accept additional attributes or properties that are utilized by the player to provide better flexibility. These configuration parameters can be set only in the configuration XML.

Common Parameters

Common Parameters are those parameters that can be set in flashvars or the configuration XML. When a parameter is set in both flashvars and the XML, **the setting in flashvars will override the setting in XML**.

For a detailed explanation of the parameters, refer to the *Configuration Parameters* section

NOTE: You can also use the *Configurator* page to generate a custom Flashvars or the configuration XML.

Configuration Parameters

The following parameters can be set using flashvars or the configuration XML (Common parameters).

NOTE: The *Required Parameters* are necessary during the initialization process of the player.

All these parameters are set in the *player* node of the configuration XML. Refer to the *XML Configuration In-depth* section for more details.

- **settings_url:**

  [Required parameter] [String] [No default setting]

  Specifies the absolute/relative URL to the configuration XML file defining core or expanded settings for the player.

- **cache_bust_key**

  [Required parameter] [Number] [No default setting]

  Setting this parameter ensures that the player loads a fresh copy of the required file every time. This parameter can be any value, but it is recommended to be a UNIX timestamp to preserve uniqueness of the whole URL. A query string parameter of *_=[timestamp]* will be added to each URL for SWF, XML, and image file.

- **controls**

  [Required parameter] [Boolean] [No default setting]

  This parameter sets the visibility of the control bar.

  true: displays the control bar.

  false: does not display the control bar.

- **auto_hide**

  [Required parameter] [Number] [No default setting]

  Sets the duration (in seconds) for the control bar to remain visible after the user has stopped interacting with the player controls. Setting this parameter to *-1* would keep the control bar visible at all times.

- **auto_play**

  [Optional parameter] [Boolean] [Default: true]

  Specifies if the player should begin playback automatically after the initialization process is complete.

  true: Starts playing the video once the player is loaded and ready.

  false: Loads the video, but, does not automatically begin playback until the user clicks on the Play/pause button or the play overlay button to begin playback.

- **auto_replay**

  [Optional parameter] [Boolean] [Default: true]

  Specifies if the player should automatically replay the video when the end of playback is reached

  true: Restarts video playback when the end of playback has been reached.

  false: When the video playback ends, a *replay* button is displayed in the player UI. Clicking on the replay button will replay the video.

- **video_url**

  [Optional parameter] [String] [No default setting]

  Sets the location of the video content

  If an incorrect location is specified, an error message is displayed within the player UI notifying the same.

- **volume**

  [Optional parameter] [Number] [Default: 80]

  Sets the initial volume of the player

  Accepted value should range between 0 and 100.

- **rewind_interval**

  [Optional parameter] [Number] [Default: 15]

  Sets the rewind duration (in seconds) when the rewind button is clicked.

- **dvr_enabled**

  [Optional parameter] [Number] [Default: 0]

  Set this parameter to enable or disable DVR functionality for a live stream.

  1: Enables DVR functionality

  0: Disables DVR functionality

- **start_bitrate_index**

  [Optional parameter] [Number] [Default: -1]

  Sets the starting index that all MBR streams should start

  -1: Would use normal switching rules applicable to MBR streams

  index value: a valid positive index value in the SMIL file.

- **captioning**

  [Optional parameter] [String] [No default setting]

  Specifies the absolute/relative location of the Timed Text (TT) file for the Closed Captioning feature

  NOTE: For more details on the Closed Captioning functionality, refer to *Working with Closed Captioning* section of this user guide

- **font_size**

  [Optional parameter] [Number] [Default: 12]

  Sets the specified font size for the Timed Text (TT) displayed in the Closed Captioning component

- **js_callback**

  [Optional parameter] [String] [Default: AEP.jsCallbackHandler]

  Specifies the JavaScript page function to invoke when notifying the page environment of standard events

- **media_analytics_logging_enabled**

  [Optional parameter] [Boolean] [Default: false]

  When this parameter is set to *true,* the log window for media analytics is displayed as a separate HTML popup. The log information displayed can be used to troubleshoot issues related to media analytics. **The parameter should only be set to *true* for troubleshooting purposes and not in the production environment.**

Sample Flashvars Configuration

```
   <script type="text/javascript">
      var swfVersionStr = "10.1.0";
      var xiSwfUrlStr = "playerProductInstall.swf";
      var flashvars
      flashvars    = {
                           auto_play : "true",
                           volume : "50",
                           video_url :
"http://mediapm.edgesuite.net/edgeflash/public/debug/assets/smil/buckbunny-vod.smil",
                           start_bitrate_index : "-1",
                           auto_replay : "true",
                           dvr_enabled : "0",
                           rewind_interval : "15",
                           settings_url: 'sample_flashvars_config.xml',
                           cache_bust_key: 123456,
```

```
                        location: window.location.host+window.location.pathname,
                        controls: true,
                        auto_hide: 2
                };

    var params ={
                        allowFullScreen: 'true',
                        allowScriptAccess: 'always',
                        bgcolor:"#000000"
                };


    swfobject.embedSWF
           ('AkamaiStandardPlayer.swf'
           , 'akamaiStandardPlayer'
           , '604', '341'
           , swfVersionStr
           , xiSwfUrlStr
           , flashvars
           , params
           , { name: 'akamaiStandardPlayer' }
    );

    swfobject.createCSS("#flashContent", "display:block;text-align:left;");
</script>
```

NOTE: In the above sample code, the SWObject would load the content into a DIV with the "Id" as "flashContent".

The *sample_flashvars.html* in the *samples* folder demonstrates configuration of the player using flashvars.

Sample XML Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
    <player
           auto_play = "true"
           volume = "100"
           data_feed_url = "resources/feeds/caption_feed.js"
           start_bitrate_index = "-1"
           auto_replay = "true"
           dvr_enabled = "1"
           rewind_interval = "15"
           captioning = "captioning text"
           js_callback = "AEP.jsCallbackHandler">
    </player>
    <controls visible="true" autoHide="-1" height="5">
           <control id="playPause" />
           <control id="rewind" />
           <control id="progress" color="#FF0000" />
           <control id="loaded" color="#00FF00" />
           <control id="scrubber" />
           <control id="cc" />
           <control id="stats" />
```

```
            <control id="volume" color="#FF0000" />
            <control id="fullscreen" />
            <control id="logo" style="width: 164px; height: 58px; left: 10px; top: 10%;
   opacity: 0.5; background-image: url('images/akamaihd_rgb.jpg');" />
      </controls>
      <plugins>
            <plugin host="osmf" type="dynamic" absolute="true"
   id="AkamaiAdvancedStreamingPlugin">http://players.edgesuite.net/flash/plugins/osmf/advan
   ced-streaming-plugin/fp10.1/v1.4/AkamaiAdvancedStreamingPlugin.swf</plugin>
            <plugin host="osmf" type="dynamic"
   id="CaptioningPlugin">resources/plugins/CaptioningPlugin.swf</plugin>
            <plugin host="osmf" type="static"
   id="OSMFCSMALoader">com.akamai.playeranalytics.osmf.OSMFCSMALoaderInfo</plugin>
      </plugins>
      <metrics>
            <vendor id="akamai">
                  <property
   key="MEDIA_ANALYTICS_BEACON">http://79423.analytics.edgesuite.net/csma/configuration/CSM
   ASampleConfiguration.xml</property>
            </vendor>
      </metrics>
   </application>
```

XML Configuration In-depth

If you examine the above XML in the ***Sample XML Configuration*** section above, you will notice that the schema contains 4 sets of main nodes or sections – ***player***, ***controls***, ***plugins*** and ***metrics***.

<player> node

```
<player
      auto_play = "true"
      volume = "100"
      video_url =
"http://mediapmt.edgesuite.net/edgeflash/public/debug/assets/smil/buckbunny-vod.smil"
      start_bitrate_index = "-1"
      auto_replay = "true"
      dvr_enabled = "1"
      rewind_interval = "15"
      captioning = "captioning text"
      js_callback = "AEP.jsCallbackHandler">
</player>
```

The attributes in the *player* node are used to set *content related parameters*. *Content related parameters* are those that directly relate to playback of video content such as ***auto_play, video_url, auto_replay*** etc. These parameters can be set in flashvars as well.

For more details on these parameters, refer to the *Configuration Parameters* section.

<controls> node

```
<controls visible="true" autoHide="-1" height="5">
      <control id="playPause" />
      <control id="rewind" />
      <control id="progress" color="#FF0000" />
      <control id="loaded" color="#00FF00" />
      <control id="scrubber" />
      <control id="cc" />
      <control id="stats" />
      <control id="volume" color="#FF0000" />
      <control id="fullscreen" />
      <control id="logo" style="width: 164px; height: 58px; left: 10px; top: 10%; opacity:
0.5; background-image: url('images/akamaihd_rgb.jpg');" />
</controls>
```

The ***Controls*** node contains individual sub-nodes for each player control. The ***visible, autoHide*** and ***height*** parameters can also be set from here. Setting these parameters in flashvars will override the setting in these attributes.

Simple color themes can be set for the ***progress, loaded, volume*** and ***logo*** controls using the ***color*** attribute by specifying the hexadecimal value for the attribute.

The ***logo*** component can also be styled based on specific requirements. Styling supports basic CSS standards as shown in the sample above.

The various style parameters supported are:

- *width*: width of the logo as a percentage or in pixels
- *height*: height of the logo as a percentage or in pixels
- *left*: the distance from the left edge of the player UI. The value can be specified in pixels or as a percentage.
- *right*: the distance from the right edge of the player UI. The value can be specified in pixels or as a percentage.
- *top*: the distance from the top edge of the player UI. The value can be specified in pixels or as a percentage.
- *bottom*: the distance from the bottom edge of the player UI. The value can be specified in pixels or as a percentage.
- *opacity:* sets the opacity of the logo. Accepted value ranges from 0 to 1 (in decimals)
- *background-image:* The absolute or relative URL of the logo image. This can be a PNG/GIF or JPEG.

A control can be removed from the control bar by removing or commenting out the related control node. When a control is removed from the control bar, the rest of the controls align automatically to fit the width of the control bar.

<plugins> node

```
<plugins>
   <plugin host="osmf" type="dynamic" absolute="true"
   id="AkamaiAdvancedStreamingPlugin">http://players.edgesuite.net/flash/plugins/osmf/advan
   ced-streaming-plugin/fp10.1/v1.4/AkamaiAdvancedStreamingPlugin.swf</plugin>
   <plugin host="osmf" type="dynamic"
   id="CaptioningPlugin">resources/plugins/CaptioningPlugin.swf</plugin>
   <plugin host="osmf" type="static"
   id="OSMFCSMALoader">com.akamai.playeranalytics.osmf.OSMFCSMALoaderInfo</plugin>
</plugins>
```

The Akamai Media Player supports two kinds of external plugins – *dynamic* and *static*. *Static* plugins use information available within the code to load external plugins. *Dynamic* plugins offer the ability to load the plug-in externally (like an actual available

SWF). The *type* attribute in the node sets *dynamic/static* property for a plugin. It is recommended to retain current settings for the plugins.

The node value of the plugin node specifies the location of the plugin. ***Unless otherwise required, it is recommended to retain the current location of the AASP and the OSMFCSMALoader***. The location of the captioning plugin can be changed depending on the file-folder structure of the player.

Supported dynamic plugins include -the ***Akamai Advanced Streaming Plugin (AASP)*** and the ***Captioning Plugin.*** The only static plugin supported by the player currently is the ***Akamai Analytics (CSMA)*** plugin.

Refer to the *About the Plugins* section for more details on the plugins used by the player.

<metrics> node

The ***metrics*** node is used to configure plugins that gather playback metrics sent from the player as per requirements laid out by each vendor. The Akamai Media Player only supports the Media Analytics plugin.

```
<metrics>
   <vendor id="akamai">
      <property
      key="MEDIA_ANALYTICS_BEACON">http://79423.analytics.edgesuite.net/csma/configuration/
      CSMASampleConfiguration.xml</property>
   </vendor>
</metrics>
```

Refer to the *Integrating with Media Analytics* section below for more detailed information on configuring media analytics.

ABOUT THE PLUGINS

Akamai Advanced Streaming Plugin

The plugin handles loading and playing of all media types over the Akamai network, including on-demand and live streaming with FMS, progressive download, HD Network for Flash, and Zeri content. This bundled approach allows the player to load only one plugin. The plugin also supports connection level and stream level authentication for FMS, and player verification for HD Network for Flash.

Captioning Plugin

The ***Captioning Plugin*** is required for the Closed Captioning functionality. The plugin loads the Timed Text (TT) displayed in the within player interface.

Media Analytics Plugin

The plugin provides business reporting solutions related to stream media usage. The plugin internalizes the external absolute SWF URL to the media analytics library and the associated beacon file used to configure the implementation of media analytics related to the player and business unit.

INTEGRATING WITH MEDIA ANALYTICS

Media Analytics provides detailed client-side and server-side reporting services. Server-side reports typically contain audience engagement and content usage information. The client-side reports provide information on user interaction and bandwidth. The Akamai Media Player supports Client-side Media Analytics (CSMA).

CSMA: Overview

CSMA support is enabled through the CSMA component – a plug-in used by the player to gather various statistics that would be used to generate the reports. The CSMA plug-in uses a configuration XML file to identify the statistics to be collected and log them in a specific location. This configuration XML file path is can be found in the *portal*.

CSMA also allows you to set *Custom Dimensions* for reports. *Custom Dimensions* would help you track any specific data passed from the player. For example, the stream name can be passed from the player and reports can be generated (in the *portal*) based on streams accessed.

CSMA: Configuring the player

The player requires – Plugin Configuration and Metrics Configuration

Plugin Configuration

The first step is to configure the CSMA *plugin.* The player uses a *static* plugin for CSMA, meaning to say that the location of the plugin cannot be configured.

More information on *plugin* configuration is available in the *Configuring the Player* section.

The *plugin* node in the player configuration XML needs to contain the plugin namespace as shown below. This would automatically load the plugin library into the player.

```
<plugin host="osmf" type="static"
id="OSMFCSMALoader">com.akamai.playeranalytics.osmf.OSMFCSMALoaderInfo</plugin>
```

Metrics Configuration

The next step would be to configure the metrics related parameters. This is done in the *metrics* node of the player configuration XML. A sample snippet is shown below:

```
<metrics>
   <vendor id="akamai">
      <property
      key="MEDIA_ANALYTICS_BEACON">http://79423.analytics.edgesuite.net/csma/configuration/
      CSMASampleConfiguration.xml</property>
   </vendor>
</metrics>
```

vendor node

The *vendor* node would be the container node for each supported metric plugin. It must be noted that the player currently supports only the Akamai Media Analytics plugin.

The *id* attribute for the *vendor* node would specify the vendor name (or id). The *id* attribute for the plugin would be *akamai*

property node

The *property* node is used to specify each configurable parameter of the plugin. Since the plugin works with *key/value* pairs, the *property* node has been structured to accept the *key* as an attribute and the value as its node value.

The first *property* node needs to specify the CSMA configuration file. This is required by the plugin to load its configuration info. The *MEDIA_ANALYTICS_BEACON* key is used to specify the beacon location, which is specified as the node value. This is shown in the example snippet above.

Custom Dimensions for Media analytics

As mentioned earlier, the Akamai Media Player also supports reporting based on custom dimensions. Custom dimensions can be passed into the player by specifying the parameters in the player configuration XML, flashvars or using the Javascript API.

NOTE: It should be noted here that the custom dimensions supported by the player needs to be configured in the *portal* before it can be passed from the player for reporting.

Using player configuration XML

Custom dimensions are specified in the *metrics* node of the player configuration XML file. A *dimensions* node is used to contain all custom dimensions passed into the player. Each custom dimension is specified in an individual *property* node. The custom dimension name is specified in the *key* attribute of the *property* node and the value is passed in as its node value. An example is shown in the code snippet below.

```
<metrics>
   <vendor id="akamai">
      <property
   key="MEDIA_ANALYTICS_BEACON">http://79423.analytics.edgesuite.net/csma/configuration/CSM
   ASampleConfiguration.xml</property>
   <dimensions default="N/A">
      <property key="customDimension" mapType="value">customDimension-value</property>
      <property key="streamUse" mapType="value">streamUse-value</property>
   </dimensions>
   </vendor>
</metrics>
```

Using flashvars

Individual parameters can be set in flashvars for passing in the custom dimensions. The player will extract the custom dimension *key* from the parameter name and the *value* would be the value specified in the flashvar. **It must be noted that custom dimension parameter names should begin with "report_" and must be in the format "report_customDimension"**. The following example shows custom dimensions being passed into the player using flashvars.

```
var flashvars =
{
      settings_url: 'sample_vod_playlist_config.xml',
      cache_bust_key: 123456,
      location: window.location.host+window.location.pathname,
      controls: true,
      auto_hide: 2,
      report_customDimension: "customDimension-value-flashVars",
      report_streamUse: "streamUse-value-flashVars",
      report_dimensionFlashvar: "dimensionFlashvar-value"
};
```

Using Javascript API

The Javascript API *loadURL()* can also be using as an way to inject custom dimensions into the player. The *loadURL* method accepts parameters of the type *object*, which can be used to specify the key/value pairs. The following example demonstrates this usage.

```
var paramsOne = {dimensions:[{key:"customOneTestOne",value:"custom1test1"},
{key:"customOneTestTwo",value:"custom1test2"}]};
function loadVideo(src,params)
{
      $("#akamaiStandardPlayer")[0].loadURL(src, paramsOne);
}
```

JAVASCRIPT API

The Adobe Flash Player has a JavaScript-Flash bridge that allows communication into and out of SWF files to allow for state changes and data to be passed back and forth at the page level. The Akamai Media Player has a robust set of External Interface JavaScript API calls that can control basic video playback functionality and deliver status information to the containing page.

NOTE: The *sample_js_console.html* page demonstrates the usage of the JS API supported by the player.

Calls made to the player can be prefixed with the player object that can be retrieved using the JavaScript *getElementById()* method or using JQuery.

Using JavaScript

```
var videoPlayer = document.getElementById("akamaiStandardPlayer");
```

Using JQuery

```
var videoPlayer = $("#akamaiStandardPlayer")[0]
```

Methods

The following methods can be used to interact with the player..

loadURL (url:String, [optional] params:object)

Loads the media resource specified in the *url* parameter. Additional parameters can also be sent to the player as an object. Typically, these parameters can be custom dimensions for media analytics

Syntax:
Loading video only:

```
videoPlayer.loadURL("http://mediapm.edgesuite.net/edgeflash/public/debug/assets/smil/buck
bunny-vod.smil")
```

Loading video and passing additional parameters:

```
var params = {dimensions:[{key:"customOneTestOne",value:"custom1test1"},
{key:"customOneTestTwo",value:"custom1test2"}]};

videoPlayer.loadURL(src, params);
```

pause()

Pauses playback of the video that is currently playing

Syntax:

```
videoPlayer.pause();
```

unpause()

Paused media will be resumed; stopped video will be started or restated.

Syntax:

```
videoPlayer.unpause();
```

stopPlayer()

Stops the video playback. This is different from pausing a video. The video content will be stopped and key events are fired in the player that will invoke specific "stopped" functionality to occur. This effectively mimics a piece of content that has played through

its duration. Note: the call is actually called "stopPlayer" due to a limitation in Adobe Flash and browser support that does not allow a simpler "stop" method name.

Syntax:

```
videoPlayer.stopPlayer();
```

mute()
Mutes all audio. The video playback will continue

Syntax:

```
videoPlayer.mute();
```

unmute()
Unmutes all audio. The video playback will continue

Syntax:

```
videoPlayer.unmute();
```

jumpToTime(seconds:number)
Seeks to the duration passed to the player as the parameter

Syntax:

```
videoPlayer.jumpToTime(30);
```

setVolume(volume:number)
Sets the current playback volume. The volume is passed to the player as a parameter.

Syntax:

```
videoPlayer.setVolume(60);
```

getVolume()
Retrieves the current playback volume

Syntax:

```
videoPlayer.getVolume();
```

currentTime()
Retrieves the elapsed time of the currently loaded video content

Syntax:

```
videoPlayer.currentTime();
```

duration()
Retrieves the total duration of the video that is currently loaded

Syntax:

```
videoPlayer.duration();
```

playbackKbps ()
Retrieves the current playback bandwidth (in kbps)

Syntax:
```
videoPlayer.playbackKbps();
```

getPlayerState()
Retrieves the string literal of the playing state of the currently loaded video content. Current player states are: playing, stopped, and paused.

Syntax:
```
videoPlayer.getPlayerState();
```

isPlayerPlaying():
Returns a true or false value indicating whether or not the player is currently in a playing state.

Syntax:
```
videoPlayer.isPlayerPlaying();
```

isStopped():
Returns a true or false value indicating whether or not the currently loaded video content is in a stopped state or not. A stopped state only occurs before a video is first played, has completely played through the duration, or encounters a problem during playback or loading that causes an OSMF stopped event to occur.

Syntax:
```
videoPlayer.isStopped();
```

isBuffering():
Returns a true or false indicating whether or not the currently loaded video content is buffering or not buffering.

Syntax:
```
videoPlayer.isBuffering();
```

Working with Events

Events are dispatched by the player from the point of initialization (when the player is ready to send/receive notifications) to the point when the video playback has ended. These events can be handled in the *jsCallbackHandler*. The *jsCallbackHandler* is configured in the player configuration. Refer to the *Configuration Parameters* section for more info. A *switch* statement can be used in JQuery or JavaScript to handle each event as shown in the sample snippet below.

```
var AEP = {
      jsCallbackHandler: function(objId, eventName, data)
      {
            switch (eventName)
            {
                  case "jsApiReady":
                        loadFeed();
                  break;
```

```
                    case "mediaPlayerPlaybackClose":
                          onEnded();
                    break;

                    case "mediaPlayerVolumeChanged":
                          if (data.volume <= 0)
                          {
                                mutedHandler();
                          }
                          else
                          {
                                unmutedHandler();
                          }
                    break;

             }
      }
};
```

Supported Events

The following events are supported by the player.

jsApiReady
This event is fired by the player when it is ready to send/receive notifications from the page.

mediaPlayerPlaybackClose
This event is dispatched when the video playback has ended.

mediaPlayerPlaying
This event is dispatched as the player continues to play the video.

mediaPlayerPaused
This event is dispatched when the user pauses the video.

mediaPlayerVolumeChanged
This event is dispatched when the user changes the player volume. It returns the *current volume* which can be utilizing by data parameter as shown in the sample snippet above.

SECURE STREAMING SUPPORT

"Secure Streaming" is an optional feature of the Akamai HD Network. This feature allows content providers to restrict unauthorized access to content – live and on-demand. The Akamai Media Player supports "TokenAuth" for all streams types and supported protocols.

Note: For Secure Streaming to work, you would need to enable the feature in the HD Network configuration.

Using Token Authentication to secure streams

"Token authentication" uses "tokens" to enable secure streaming. Basically, tokens contain information which are read and validated by the Akamai Edge Servers, based on predefined authorization criteria. Tokens contain a profile and password that uniquely identify the content provider as the valid creator of the token.

A Token needs to be generated and implemented for streams that need to be secured. The token generation process is different for HDN and RTMP-based streams. The table below provides the details to generate a token for each stream type.

<table>
<tr><th colspan="2">RTMP</th></tr>
<tr><td>SBR</td><td>

Steps:

- **Generate the token:** Use the token generation script (links available in *Token Generation Script* section – *Token Generation Script for RTMP streams*) to generate the tokenized URL by passing the following parameters:
  - **Token type**: For securing RTMP streams, this needs to be "d" type tokens
  - **Path**: location of the media file.
  - **Profile**: Name that identifies the configuration
  - **Password**: Shared secret between Akamai and the content provider
  - **Window**: defines a span of time when access to the stream is enabled. This needs to be specified in the UNIX Epoch format

Example usage in PHP:

```
<?PHP
$factory = new StreamTokenFactory;
$userKey = file_get_contents("my_user_key.bin");

$token = $factory->getToken("d",
"cp100125.edgefcs.net/ondemand/mp4:vids2/backCountry_HD.mp
4",
NULL,
"my_profile",
"my_passwd",
NULL,
86400,
NULL,
NULL,
NULL);

echo "Token: " . $token->getToken() . "\n";
?>
```

- **Create the tokenized URL:** Once the token is generated, the next step is to append the generated token with the SBR stream that is secured. For example, if your stream URL is:
  `rtmp://cp100125.edgefcs.net/ondemand/mp4:vids2/backCountry`

</td></tr>
</table>

	`_HD.mp4`
	
	Your tokenized URL would be:
	`rtmp://cp100125.edgefcs.net/ondemand/mp4:vids2/backCountry_HD.mp4?auth=xxx&aifp=v001`
	
	• **Pass the tokenized URL to the player:** The tokenized URL can now be passed to the player. This can be done using the *video_url* parameter in flashvars or the player configuration XML. It can also be passed into the player using the Javascript API – *loadURL()* method.

MBR	*Steps:*
	• **Generate the token:** Use the token generation script (links available in *Token Generation Script* section - – *Token Generation Script for RTMP streams*) to generate the tokenized URL by passing the following parameters:
	o **Token type**: For securing RTMP streams, this needs to be "d" type tokens
	o **Path**: the slist for all bitrates in the SMIL file.
	o **Profile**: Name that identifies the configuration
	o **Password**: Shared secret between Akamai and the content provider
	o **Window**: defines a span of time when access to the stream is enabled. This needs to be specified in the UNIX Epoch format
	
	Example usage in PHP:
	`<?PHP`
	`$factory = new StreamTokenFactory;`
	`$userKey = file_get_contents("my_user_key.bin");`
	
	`$token = $factory->getToken("d",`
	`"vids2/princeOfPersia_256.mp4;vids2/princeOfPersia_512.mp4;vids2/princeOfPersia_1000.mp4;",`
	`NULL,`
	`"my_profile",`
	`"my_passwd",`
	`NULL,`
	`86400,`
	`NULL,`
	`NULL,`
	`NULL);`
	
	`echo "Token: " . $token->getToken() . "\n";`
	`?>`
	
	• **Create the tokenized URL:** Once the token is generated, the next step is to append the generated token with the SBR stream that is secured. For example, if your stream URL is:
	`http://mysever.edgesuite.net/princeOfPersia_rtmp.smil`
	
	Your tokenized URL would be:
	`http://mysever.edgesuite.net/princeOfPersia_rtmp.smil?auth=xxx&aifp=v001`
	
	• **Pass the tokenized URL to the player:** The tokenized URL can now be passed to the player. This can be done using the *video_url* parameter in flashvars or the

	player configuration XML. It can also be passed into the player using the Javascript API – *loadURL()* method.
HDN 1.0 for Flash	

SBR	*Steps:*
	• **Generate the token:** Use the token generation script (links available in *Token Generation Script* section – *Token Generation Script for HDN 1.0 streams*) to generate the tokenized URL by passing the following parameters:
	o **Path**: location of the media file.
	o **Key name**: A string to contain the auth code.
	o **Window**: defines a span of time when access to the stream is enabled. This needs to be specified in the UNIX Epoch format
	o **Salt**: Shared secret between Akamai and the content provider
	
	Example usage in PHP:
	`<?PHP`
	`$sUrl = "/mp4s/princeOfPersia_256.mp4";`
	`$sParam = "primaryToken";`
	`$nEventDuration = 86400;`
	`$nWindow = time();`
	`$sSalt = "akamai123";`
	
	`$token = urlauth_gen_url($sUrl, $sParam, $nWindow, $sSalt, $sExtract, $nTime);`
	`echo "var token = '".$token."';";`
	`?>`
	
	• **Create the tokenized URL:** Once the token is generated, the next step is to append the generated token with the SBR stream that is secured. For example, if your stream URL is:
	`http://tokenauth-f.akamaihd.net/mp4s/princeOfPersia_256.mp4`
	
	Your tokenized URL would be:
	`http://tokenauth-f.akamaihd.net/mp4s/princeOfPersia_256.mp4?primaryToken=xxxxxxxx`
	
	• **Pass the tokenized URL to the player:** The tokenized URL can now be passed to the player. This can be done using the *video_url* parameter in flashvars or the player configuration XML. It can also be passed into the player using the Javascript API – *loadURL()* method.

MBR	*Steps:* • **Create the CSMIL:** For MBR streams, you need to first create an in-URL CSMIL. This needs to be created manually. Refer to the *Creating the CSMIL* section below this table for info • **Generate the token:** Use the token generation script (links available in *Token Generation Script* section – *Token Generation Script for HDN 1.0 streams*) to generate the tokenized URL by passing the following parameters: o **Path**: the in-URL CSMIL. o **Key name**: A string to contain the auth code. o **Window**: defines a span of time when access to the stream is enabled.

	This needs to be specified in the UNIX Epoch format
	o **Salt**: Shared secret between Akamai and the content provider
	
	Example usage in PHP:
	`<?PHP`
	`$sUrl = /mp4s/princeOfPersia_,256,512,1000,.mp4.csmil/bitrate=0";`
	`$sParam = "primaryToken";`
	`$nEventDuration = 86400;`
	`$nWindow = time();`
	`$sSalt = "akamai123";`
	
	`$token = urlauth_gen_url($sUrl, $sParam, $nWindow, $sSalt, $sExtract, $nTime);`
	`echo "var token = '".$token."';";`
	`?>`
	
	• **Create the tokenized URL:** Once the token is generated, the next step is to append the generated token with the SBR stream that is secured. For example, if your stream URL is:
	`http:// myserver.edgesuite.net/princeOfPersia.smil`
	
	Your tokenized URL would be:
	`http://myserver.edgesuite.net/princeOfPersia.smil?primaryToken=xxxxx`
	
	• **Pass the tokenized URL to the player:** The tokenized URL can now be passed to the player. This can be done using the *video_url* parameter in flashvars or the player configuration XML. It can also be passed into the player using the Javascript API – *loadURL()* method.

Token Generation Script

Token generation script for RTMP streams:

The sample code to generate tokenized URL can be downloaded from the following links:

- **ASP**: http://download.akamaitools.com.edgesuite.net/auth/SecureStreaming/SecureStreamingASP-3.0.2.zip
- **C Sharp**: http://download.akamaitools.com.edgesuite.net/auth/SecureStreaming/SecureStreamingCSharp-3.0.2.zip
- **Java**: http://download.akamaitools.com.edgesuite.net/auth/SecureStreaming/SecureStreamingJava-3.0.2.zip
- **Perl**: http://download.akamaitools.com.edgesuite.net/auth/SecureStreaming/SecureStreamingPerl-3.0.zip
- **PHP**: http://download.akamaitools.com.edgesuite.net/auth/SecureStreaming/SecureStreamingPHP-3.0.1.zip

Token generation scripts for HDN 1.0 streams

The token generation script can be downloaded from Akamai EdgeControl by navigating to:

- *My Services > HTTP Downloads > Tools*
- From the Tools page, select the "*Token Generators for Access Control (by IP or URL)"* link available in the "Development Tools" section
- From the "Token Generators" page, download the script based on your development platform. The download links are available in the *URL-based* section. Scripts are available for ASP, PHP, Perl, Java, C#, and C

Creating the CSMIL

CSMIL is a set of in-URL parameters – filename of the stream, the bitrates contained in the SMIL, the file extension. To construct an in-URL CSMIL, the following parameters are required

- **Path:** location of the media file without the file extension.
  For example: http://tokenauth-f.akamaihd.net/mp4s/princeOfPersia
- **Bitrates:** various bitrates specified in the SMIL file as per its index

Example of a SMIL file:

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
"http://www.w3.org/2001/SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
          <head>
                <meta name="title" content="Price Of Persia Trailer" />
                <meta name="httpBase" content="http://tokenauth-f.akamaihd.net" />
                <meta name="vod" content="true" />
          </head>
          <body>
          <switch>
               <video src="mp4s/princeOfPersia_256.mp4" system-bitrate="250000"/>
               <video src="mp4s/princeOfPersia_512.mp4" system-bitrate="600000"/>
               <video src="mp4s/princeOfPersia_1000.mp4" system-bitrate="4000000"/>
          </switch>
          </body>
</smil>
```

In the above example, the bitrates would be to 256, 512 and 1000.

- **File extension:** The file extension of the . For example: .mp4
- **CSMIL extension:** Adding the .CSMIL file extension signifies the in-URL CSMIL
- **/bitrate=0:** First bitrate index in the SMIL file

Using the above parameters in the format:
<subdirectory>/prefix,bitrate1,bitrate2,bitrate3,extension.csmil/bitrate=0

The in-URL CSMIL would be:

```
http://tokenauth-
f.akamaihd.net/mp4s/princeOfPersia_,256,512,1000,.mp4.csmil/bitrate=0
```

WORKING WITH CLOSED CAPTIONING

This version of the Akamai Media Player supports *Closed Captioning (CC)*. CC is a process of displaying transcriptions related to the visuals displayed in the player. The Akamai Media Player uses a *Captioning Plugin* (as discussed in previous sections) to implement this feature. This feature can be enabled and configured using flashvars/configuration XML.

Generating the Timed Text (TT)

A Timed Text XML file needs to be generated for each video that would have the CC option enabled. The captioning plugin loads and parses the TT file to display the captions in the player UI at the specific duration also defined in the TT file. The player supports standards laid out by the World Wide Web Consortium (W3C).

The TT file can be generated using third-party tools such as *Caption Wrap* or *Hi-Caption Studio*. The content of a sample TT file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
xmlns:tts="http://www.w3.org/2006/04/ttaf1#styling">
	<head>
		<styling>
			<style id="emph" tts:fontStyle="italic" tts:fontWeight="bold"/>
			<style id="quote" tts:fontStyle="italic"/>
		</styling>
	</head>
	<body>
		<div>
			<p begin="00:00:11">Building on a decade of innovation.</p>
			<p begin="00:00:13">One company remains uniquely capable of <span
style="emph">powering the Internet</span>,</p>
			<p begin="00:00:17">having forever changed the World Wide Web.</p>
			<p begin="00:00:21">Founded commercially just ten years ago,</p>
			<p begin="00:00:23">Akamai&#39;s origins date back to the mid 1990&#39;s
inside the halls of MIT.</p>
			<p begin="00:00:29">An academic project looking for ways to reduce
Internet traffic congestion</p>
			<p begin="00:00:34">set in motion a Web revolution.</p>
			<p begin="00:00:36">Professor Tom Leighton and graduate student Danny
Lewin,</p>
			<p begin="00:00:39">working with fellow mathematicians and computer
scientists,</p>
			<p begin="00:00:42">came up with a way to solve the Internet&#39;s <span
style="emph">hot spot</span> problem.</p>
			<p begin="00:00:46">This was the challenge.</p>
			<p begin="00:00:48">Just as something on the Net became popular,</p>
			<p begin="00:00:50">few could access it because Web sites became
overloaded.</p>
			<p begin="00:00:54">Akamai would solve this problem.</p>
			<p begin="00:00:56">The founding team explored radical, new approaches
to changing</p>
			<p begin="00:01:00">the way Internet content was delivered</p>
		</div>
	</body>
</tt>
```

Player Configuration

The following parameters for closed captioning can be set in flashvars or the configuration XML. Optionally, you can also use the Javascript API to load the TT and video URL.

Fig: 19 – Closed captioning

Basically, there are 3 parameters you need to set in the configuration –

- ***video_url:*** URL to the video content. This need not be set for JavaScript usage
- ***caption:*** absolute or relative location of th Timed Text XML file
- ***font_size:*** font size of the caption text (in pixels)

Configuration using flashvars

```
var flashvars =
            {
                  video_url:
                  rtmp://cp67126.edgefcs.net/ondemand/mediapm/osmf/content/test/akamai_10_
                  year_f8_512K',
                  settings_url: 'resources/conf/config.xml',
                  auto_play: 'true',
                  cache_bust_key: 123456,
                  location: window.location.host+window.location.pathname,
                  captioning:
                  http://mediapm.edgesuite.net/osmf/content/test/captioning/akamai_sample_
                  caption.xml',
                  font_size: 15,
                  controls: true,
                  auto_hide: 2
            };
```

Configuration using XML

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
	<player
		video_url =
"rtmp://cp67126.edgefcs.net/ondemand/mediapm/osmf/content/test/akamai_10_year_f8_512K"
		auto_play = "true"
		volume = "50"
		start_bitrate_index = "-1"
		auto_replay = "true"
		dvr_enabled = "0"
		rewind_interval = "15"
		captioning =
"http://mediapm.edgesuite.net/osmf/content/test/captioning/akamai_sample_caption.xml"
		font_size="15"
		js_callback = "AEP.jsCallbackHandler">
	</player>
	<controls visible="true" autoHide="5" height="5">
		<control id="playPause" />
		<control id="rewind" />
		<control id="progress" color="#FF0000" />
		<control id="loaded" color="#00FF00" />
		<control id="scrubber" />
		<control id="cc" />
		<control id="stats" />
		<control id="volume" color="#0000FF" />
		<control id="fullscreen" />
		<control id="logo" style="width: 150px; height: 53px; left: 5px; top: 5px;
opacity: 0.5; background-image: url('../images/akamai_logo.png');" />
	</controls>
	<plugins>
		<plugin host="osmf" type="dynamic" absolute="true"
id="AkamaiAdvancedStreamingPlugin">http://players.edgesuite.net/flash/plugins/osmf/advanced-
streaming-plugin/fp10.1/v1.3/AkamaiAdvancedStreamingPlugin.swf</plugin>
		<plugin host="osmf" type="dynamic"
id="CaptioningPlugin">../resources/plugins/CaptioningPlugin.swf</plugin>
		<plugin host="osmf" type="static"
id="OSMFCSMALoader">com.akamai.playeranalytics.osmf.OSMFCSMALoaderInfo</plugin>
	</plugins>
	<metrics>
		<vendor id="akamai">
			<property
key="MEDIA_ANALYTICS_BEACON">http://79423.analytics.edgesuite.net/csma/configuration/CSMASam
pleConfiguration.xml</property>
		</vendor>
	</metrics>
</application>
```

Using JavaScript API

```
var src =
"rtmp://cp67126.edgefcs.net/ondemand/mediapm/osmf/content/test/akamai_10_year_f8_512K"
```

```
var params = {
captioningUrl:"http://mediapm.edgesuite.net/osmf/content/test/captioning/akamai_sample_captio
on.xml"};

function loadVideo(src,params)
{
      $("#akamaiStandardPlayer")[0].loadURL(src, params);
}
```

NOTES:

1. Refer to the *Configuring the Player* section for more info on player configuration
2. The *sample_accessibility.html* page in the *samples* folder demonstrates the accessibility feature and configuration. Examine the *sample_accessibility_config.xml* file for configuration settings
3. Font size can't be specified using Javascript

To load the plugin, the configuration XML needs to be updated as well. The ***plugins*** node in the configuration XML should contain the node containing the location of the *Captioning Plugin* as shown below. The default configuration XML in the player package already contains the relevant node

```
<plugin host="osmf" type="dynamic"
id="CaptioningPlugin">resources/plugins/CaptioningPlugin.swf</plugin>
```

Once the configuration is complete, the player will display the captions at each cue point.

PLAYER PACKAGE AND DEPLOYMENT

Pre-requisites for deployment

To use the player, you need:

- A location to host the player—origin or a NetStorage account with an Akamai HTTP delivery configuration
- Flash compatible media assets
    - Supported video codecs
        - FLV: On2 VP6, Sorenson Spark, H.264
        - MP4: H.264
    - Supported audio codecs
        - FLV: MP3, ADPCM, Nellymoser, Speex, AAC
        - MP4: AAC, MP3
    - Supported delivery methods – Progressive Download (PDL), Streaming – Video On-demans (VOD) and Live Streaming
    - Supported standards – SMIL, RSS Playlists

System requirements

The user's computer should have the following minimum configuration.
Processor: 1Ghz
RAM: 1GB
Screen resolution: 1024x768 (normal screen layout), 1280x720 (widescreen layout)
Video RAM: 512 MB
Flash plugin: 10.1+
Browser: Microsoft Internet Explorer 7+, Firefox 3.6+, Chrome 3+
Operating system: Microsoft Windows XP SP3+/Vista/7, Mac OS X

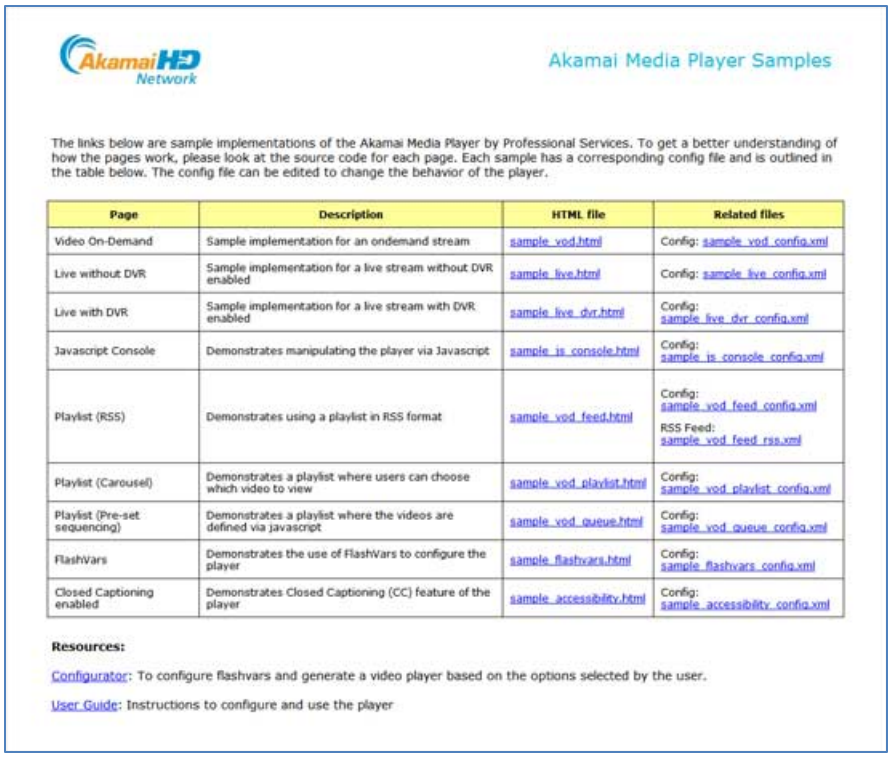
Player package

The player is distributed as a ZIP file package. The ZIP file should be extracted into a web hosted directory. The package contains the following folders and files:

File/Folder name	Description
configurator	Folder containing the Player Configurator page and the resources used by the same
css	Folder containing CSS files used by the testharness pages
images	A common folder containing the images used by the player and the HTML pages
js	A common folder containing the Javascript files used by the player and HTML pages
resources	Folder containing all the resources used by the player
samples	Contains the sample pages to demonstrate various implementations of the player
AkamaiStandardPlayer.swf	The compiled version of the player
AkamaiStandardPlayer-debug.swf	The debug version of the player to be used for troubleshooting
index.html	Start page for Samples. Provides links to all samples contained in the Samples folder

Samples

The *samples* folder contains various sample implementations of the player. Examine the HTML page source and source of associated files to get familiar with the specific implementations. The *index.html* page contains links to each sample and associated configuration XML.



Akamai Media Player Samples

The links below are sample implementations of the Akamai Media Player by Professional Services. To get a better understanding of how the pages work, please look at the source code for each page. Each sample has a corresponding config file and is outlined in the table below. The config file can be edited to change the behavior of the player.

Page	Description	HTML file	Related files
Video On-Demand	Sample implementation for an ondemand stream	sample_vod.html	Config: sample_vod_config.xml
Live without DVR	Sample implementation for a live stream without DVR enabled	sample_live.html	Config: sample_live_config.xml
Live with DVR	Sample implementation for a live stream with DVR enabled	sample_live_dvr.html	Config: sample_live_dvr_config.xml
Javascript Console	Demonstrates manipulating the player via Javascript	sample_js_console.html	Config: sample_js_console_config.xml
Playlist (RSS)	Demonstrates using a playlist in RSS format	sample_vod_feed.html	Config: sample_vod_feed_config.xml RSS Feed: sample_vod_feed_rss.xml
Playlist (Carousel)	Demonstrates a playlist where users can choose which video to view	sample_vod_playlist.html	Config: sample_vod_playlist_config.xml
Playlist (Pre-set sequencing)	Demonstrates a playlist where the videos are defined via javascript	sample_vod_queue.html	Config: sample_vod_queue_config.xml
FlashVars	Demonstrates the use of FlashVars to configure the player	sample_flashvars.html	Config: sample_flashvars_config.xml
Closed Captioning enabled	Demonstrates Closed Captioning (CC) feature of the player	sample_accessibility.html	Config: sample_accessibility_config.xml

Resources:

Configurator: To configure flashvars and generate a video player based on the options selected by the user.

User Guide: Instructions to configure and use the player

Fig: 20 - *index.html* page containing links to various samples

Player Configurator

The distribution package also contains the *Player Configurator* located in the *configurator* folder. The page provides as a simple GUI to generate flashvars or the configuration XML parameters based on your inputs. The page also provides features to copy generated code to the clipboard and POST the configuration XML to NetStorage. The functionality can be modified to POST to a backend script – ASP.NET/PHP/ColdFusion page as well.

The *index.html* page also provides the direct link to the *Player Configurator* page.

Fig: 21 - The *Player Configurator* page

Deploying the player

Follow the steps below to deploy the player:

1. Extract the contents of the player distribution package.
2. Copy the following required files and folders to your hosting environment
    a. *AkamaiStandardPlayer.swf*
    b. *resources* folder
    c. *js* folder
    d. *images* folder
3. Embed the player SWF (AkamaiStandardPlayer.swf) to your container page. This can be done using HTML <OBJECT> <EMBED> tags or the SWFObject. It is recommended to use SWFObject to embed the player. You will find the *swfobject.js* file in the *js* folder.

    The code sample below provides you an example:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Akamai Media Player</title>
<script type="text/javascript" src="../js/swfobject.js"></script>
        <script type="text/javascript">
            <!-- For version detection, set to min. required Flash Player version, or 0 (or
0.0.0), for no version detection. -->
            var swfVersionStr = "10.1.0";
            var xiSwfUrlStr = "../playerProductInstall.swf";
            var flashvars = {
```

```
                        settings_url: 'sample_vod_config.xml',
                        cache_bust_key: 123456,
                        location: window.location.host+window.location.pathname,
                        controls: true,
                        auto_hide: 2
                        };

                var params = {
                        allowFullScreen: 'true',
                        allowScriptAccess: 'always',
                        bgcolor:"#000000"
                        };

          swfobject.embedSWF
           (' AkamaiStandardPlayer.swf'
           , 'akamaiStandardPlayer'
           , '604', '341'
           , swfVersionStr
           , xiSwfUrlStr
           , flashvars
           , params
           , { name: 'akamaiStandardPlayer' }
          );

</script>
</head>
<body>
<div id="akamaiStandardPlayer">
      <p>To view this page ensure that Adobe Flash Player version 10.1.0 or greater is
installed. </p>
      <script type="text/javascript">
            var pageHost = ((document.location.protocol == "https:") ? "https://" :
      "http://");
            document.write("<a href='http://www.adobe.com/go/getflashplayer'><img src='"
                                                              + pageHost +
"www.adobe.com/images/shared/download_buttons/get_flash_player.gif' alt='Get Adobe Flash
player' /></a>" );
      </script>
</div>
</body>
</html>
```

4. If required, edit the configuration XML located in the *resources/conf* folder. Refer to the *Configuring the Player* section for more info on the configuration parameters.
5. Test the player to verify intended functionality before pushing it to a production environment

TROUBLESHOOTING

The video player supports several debugging options that allow you to effectively troubleshoot any issues surrounding the player experience.

The *AkamaiStandardPlayer-debug.swf* is provided in the player package. This file needs to be embedded into a container page just like the main player SWF file (*AkamaiStandardPlayer.swf)* before using the debug tools described below. You can temporarily replace the *AkamaiStandardPlayer.swf* in your staging environment to troubleshoot any issues.

NOTE: It cannot be overstated that the first line of defense used when tracking down issues is to review the configuration file

Debug consoles

Flash Debug Player

In order to effectively debug any issues using trace or logging information within a debug version of the video player, you need to download and install the debug version of the Flash Player. Right-click on any SWF embedded in a web page to see the "Debugging" context-menu item that indicates that the Flash Debug Player is installed and ready to use. The debug player should be used for all testing, as it will allow you to diagnose issues within the player, including helpful trace and logging output and errors that may occur. The Adobe Flash Debug Player can be downloaded from the Flash Player page at adobe.com.

FlashTracer

FlashTracer is a FireFox extension for Mac and Windows that allows trace output to be displayed in a separate window or sidebar. This plugin is an effective way to quickly look into output and diagnose problems. Any developer or business user can effectively use this tool to manage testing. See the FlashTracer site for more information and documentation (http://www.sephiroth.it/firefox/flashtracer/).

Arthropod

One effective debugging tool is the Arthropod AIR application that works with side the Flash Debug Player. There is an internal debug password that needs to be added to the settings of this application in order for the debug output from the debug version of the video player to be displayed. This tool works similarly to the FlashTracer, but it is output to a separate AIR application and is secured with a special password. There is also color-coded output for the different debug warnings and/or errors that display.

The AIR application is available at: http://arthropod.stopp.se/

De MonsterDebugger

De MonsterDebugger is probably the most effective Flash Debugger available. It has many more features than the Arthropod solution. It is useful for all types of business users, but it is geared towards developers, due to its advanced layout and feature set. The De MonsterDebugger allows you to peer into the actual values of variables and objects used within the Flash application. It has debugging tools that are very similar to the Adobe Flash Builder standalone application.

The AIR application is available at: http://demonsterdebugger.com/

Troubleshooting common issues

Here is a list of common issues and steps to troubleshoot them.

Player not loading/UI is not visible

The player can be embedded using the SWFObject. This is a recommended way to embed the player on your webpage. If your webpage uses a complex nested layout, make sure the DIV that contains the player is accessible in your code. Also, make sure that the location of the player SWF is correct.

You can also check if the player has loaded by right-clicking (Win) or CTRL+click (Mac). This would display the player's context-menu which provides player related details such as build number, plugin versions etc. The context-menu would contain an item – "Movie not loaded" if the SWF is not loaded.

Player loads but control bar is not available

The possible causes for this are:

1. The player is not able to access the player configuration XML file. This can be set in the *settings_url* parameter in your flashvars. The player needs this configuration to load visibility for all UI controls.
2. The *controls* parameter in the flashvars/configuration XML is set to *false*

Stream not playing

The player displays error messages within its interface when stream related errors occur. Typical stream related errors are "Stream not found" or "Connection timeouts" are handled internally.

1. Check the player configuration XML or flashvars to make sure that the location of the stream is correct
2. Use a web debugging proxy like Charles to check if the player is connecting to the stream location
3. The Akamai support players can also be used to check stream playback issues. Bookmark the following links that would navigate you to the support players:
    a. RTMP – http://support.akamai.com/flash/
    b. HDN 1.0 for Flash - http://mediapm.edgesuite.net/edgeflash/public/debug/HDPlayer.html
4. If the stream is a LIVE stream, make sure that the Entry Point is configured correctly and the stream is being published.

Verify Analytics data from the player

The first step would be to verify if the media analytics configuration is correct. The *metrics* node in the configuration XML needs to contain the beacon path. For more information on this topic,refer to the *Integrating with Media Analytics* section.
If the configuration is correct, the next step would be to verify if the player is able to access the beacon. To verify:

1. Set the *media_analytics_logging_enabled* parameter in flashvars or configuration XML to *true*.
2. Launch the player
3. A HTML popup window containing the log would be displayed.
4. Examine the log to verify that there are no errors displayed and that the beacon query string contains all the valid parameters.

Optionally, you can also use a web proxy client like *Charles* or *Fiddler* to verify that the beacons can be accessed and data is being passed from the player.