VIDEOMETRIX®

Streaming Tag for JavaScript

API Implementation Guide

FOR FURTHER INFORMATION, PLEASE CONTACT:

comScore, Inc.

Tag Support

+1 866 276 6972

tagsupport@comscore.com

DOCUMENT VERSION 1.1401.28

Contents

Implementation Quick Start

This section contains abridged implementation instructions for advanced users.

1. Include the JavaScript library in your web page (see 🔗 Include the Library on page 17).
2. Create a new `StreamSense` object instance for each media player on the web page(1) and initialize it with your base measurement URL (see 🔗 Initialization on page 17 and 🔗 Base Measurement URL on page 8):

   ```
   var streamsense = new ns_.StreamSense({}, 'http://b.scorecardresearch.com/p?c1=2&c2=1234567');
   ```
3. Before any start of playback, set the current Stream Sense playlist (see 🔗 Set the Current Playlist on page 18):

   ```
   streamSense.setPlaylist();
   ```
4. Before each segment starts (when content changes) set the current clip (see 🔗 Set the Current Clip on page 18):

   ```
   streamSense.setClip(clipAttributes);
   ```
5. Notify of player state changes (see 🔗 Notifying of Player State Changes on page 19):
   - when the media player **starts buffering**:

     ```
     streamSense.notify(ns_.StreamSense.PlayerEvents.BUFFER, {}, position);
     ```
   - when **playback is paused** or **user starts seeking during playback** (i.e., the player is at the old position):

     ```
     streamSense.notify(ns_.StreamSense.PlayerEvents.PAUSE, {}, position);
     ```
   - when **playback starts or resumes** or **seeking is completed** (i.e., the player is at the new position):

     ```
     streamSense.notify(ns_.StreamSense.PlayerEvents.PLAY, {}, position);
     ```
   - when **playback ends**:

     ```
     streamSense.notify(ns_.StreamSense.PlayerEvents.END, {}, position);
     ```

After successfully following these steps you will have implemented the comScore Streaming Tag. Your media player should now send measurements whenever it is started. Please refer to 🔗 Testing the Implementation on page 23 for instructions on testing the implementation.

(1) This code sample assumes you have a reference `streamSense` available.

Introduction

Tagging

A *tag* is a piece of scripting or markup that is placed on a website or another web based content asset. Tags are sometimes referred to as tracking pixels or beacons. They are used to measure the consumption of digital content by an end-user. *Tagging* is the process of adding a comScore tag – also called *measurement code* or *SDK*[(2)] – to your digital content. Each time a tagged piece of content – e.g., a web page, a video stream or an application view in a (mobile) application – is used by an end-user, the tag sends data via an HTTP request to comScore's data collection servers.

Streaming Tag

The comScore Streaming Tag, used for tagging streams, incorporates accurate and comprehensive online audio and video analytics functionality. The Streaming Tag enables comScore to receive measurement insights critical to answering questions about audio and video stream usage, including advertising messages.

This documentation is equally applicable to both video and audio streaming measurement. Please ensure that you clarify with your comScore account team what type of media you should be implementing this tag onto (video and/or audio). Please do not implement this tag onto media types other than those you have been instructed to by your comScore account team.

The Streaming Tag can be an SDK with a code library or an easy to use plugin. Streaming Tags collect their data from the media player.

Streaming Tag Implementation Approaches

There are two kinds of implementations for Streaming Tags:

1. **Implementations that use a standardized, comScore-provided adapter or plugin.**
   The adapter or plugin is capable of translating internal activity inside the media player into calls to the API of the Streaming Tag library.
2. **Implementations where the developer of the player implements calls to the internal API of the Streaming Tag library.**
   These implementations do not use a standardized, comScore-provided adapter or plugin.

The implementation instructions in this document cover the second approach. These instructions allow your media player developer to use the Streaming Tag library's API methods to tag your video content with a Streaming Tag. These instructions and the JavaScript code library are intended to be used with media players which support a JavaScript API and run in a web browser or a web browser-like environment, like net-connected TV sets.

---

(2) SDK is an acronym for Software Development Kit which is typically a set of tools and documentation for the creation of software.

If your media player does not have a JavaScript API or is designed to run in another environment then please inform your client account team or contact comScore Tag Support (contact details can be found on the front page of this document). Your client account team and comScore Tag Support can confirm whether or not the comScore Streaming Tag can be used with your development environment.

Implementation Preparation

Terminology

This document uses some terms that might need to be clarified.

Stream/clip

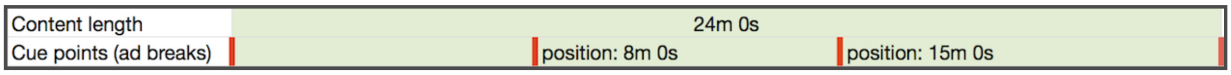
A *stream* – or *clip* – is a content asset or an advertisement that will be played by the media player.
Important: For video or audio streams that have mid-roll advertisements breaking them apart, the entire content is considered to be a discrete stream/clip.

Playlist

A *playlist* is an ordered collection of *streams* that will be played by the media player without required interference from the user. The clip number identifies streams in a playlist.

Each asset, be it content or advertisement, should be mapped to a clip in the implementation of the Streaming Tag.

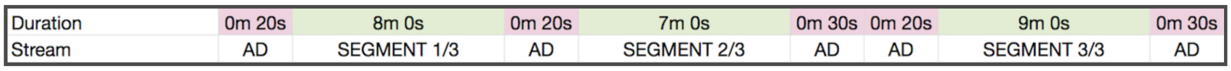
Please study the following context as an example:



The media player has loaded a long-form content stream as the content asset. The content asset has a length of 24 minutes. There are 4 cue points for ad breaks defined for this content (the red markers in the image shown above):

1. *a pre-roll at the beginning of the content (position 0:00)*
2. *mid-roll 1 at position 08:00*
3. *mid-roll 2 at position 15:00*
4. *a post-roll at the end of the content (position 24:00)*

During playback the media player loads one or more ads at the ad breaks. For a particular playback scenario the media player's playback timeline could like this:



Duration	0m 20s	8m 0s	0m 20s	7m 0s	0m 30s	0m 20s	9m 0s	0m 30s
Stream	AD	SEGMENT 1/3	AD	SEGMENT 2/3	AD	AD	SEGMENT 3/3	AD

In our example playback scenario the ad breaks for pre-roll, mid-roll 1 and post-roll all contain 1 advertisement. The ad break for mid-roll 2 contains 2 advertisements. The mid-roll ad breaks divide the content into 3 *parts*, or *segments*.

Each asset – advertisement or content – is represented by a clip. This context is translated into the following playlist with 6 clips:

Clip number	Description
1	pre-roll advertisement
2	content asset
3	mid-roll 1 advertisement

Clip number	Description
4	mid-roll 2 advertisement 1
5	mid-roll 2 advertisement 2
6	post-roll advertisement

In a Streaming Tag implementation the clips are represented as a collection of attributes or *labels*. This is implemented using objects. When your media player loads a content asset or advertisement for playback the Streaming Tag should be instructed to switch to the corresponding clip. When your media player changes back to playing the content asset after playing a mid-roll advertisement the Streaming Tag should be instructed to switch back to the clip that corresponds with the content asset.

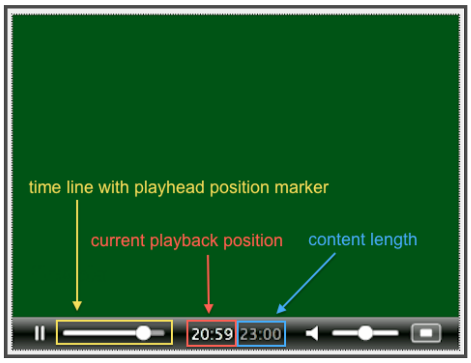
One approach to implement the playlist and clips in your media player is to maintain an array of objects that represents the current playback timeline. Clips can be added to the array as their corresponding assets are played by the media player. If the player plays an asset that was played before then the existing array element can be reused.

The setup of clips and switching between clips is discussed in Set the Current Clip on page 18. Throughout this document the context and playback scenario described above are used in the examples.

If you are unsure how to define playlists and clips, please contact your comScore account team who can help you decide on an appropriate structure.

Current Playback Position

The *current playback position* is the amount of time elapsed through the content. Often media players have a display that shows the current playback position to the end user. An example is shown in the following screenshot, where the current playback position is 20 minutes and 59 seconds:


time line with playhead position marker
current playback position
content length
20:59
23:00

The current playback position is a vital piece of information in any Streaming Tag implementation. Your media player should be able to provide the current playback position to allow for proper implementation and correct measurement.

Please make sure you are able to provide the current playback position as a value **in milliseconds** in the Streaming Tag implementation. If your player reports positions in seconds then please multiply any values by 1000 as necessary.

Base Measurement URL

Stream Sense Tags send their collected data as HTTP Get requests. To work properly, the Stream Sense Tag needs to be provided with a *base measurement URL*. The plugin will append any collected data to the base measurement URL in the HTTP requests it generates.

Please use the following URL – **replacing `1234567` with your comScore client ID** – to determine your base measurement URL:

- `http://b.scorecardresearch.com/p?c1=2&c2=1234567`

If – for example – your comScore client ID is “1000001” then the base measurement URL you should supply in the initialization is:

- `http://b.scorecardresearch.com/p?c1=2&c2=1000001`

The code examples in this document use `http://b.scorecardresearch.com/p?c1=2&c2=1234567` as the base measurement URL. Please make sure you always use your own base measurement URL to make sure your measurement data is collected properly.

Collected Data

Adding additional collected data is as easy as adding attributes to clips. The use of clips is addressed in Set the Current Clip on page 18.

The following table summarizes the mandatory attributes that are associated with each clip. These attributes are further explained on subsequent pages. The sample code in the implementation instructions will also show how to add these attributes as needed.

Clip Attributes

Attribute	Name	Description	Format	Mandatory?	Example
`ns_st_cn`	Clip Number	Determines the order of clips in a playlist	Integer	mandatory	`1`
`ns_st_ci`	Content ID	Content or asset ID. Your internal **unique** identification of each content asset[3]. This ID needs to be as unique as possible across all your content.	String	mandatory	`137849`
`ns_st_pn`	Part number	Part (or *segment*) number. Identifies a segment of the content. Should increment after a mid-roll ad break.	Integer	mandatory	`1`
`ns_st_tp`	Total parts	Total number of parts. Identifies a segment of the content. Use value `0` if you do not know the total number of parts.	Integer	mandatory	`3`
`ns_st_cl`	Clip Length	Length of the content or ad **in milliseconds**	Integer	mandatory	`1440000`[4]
`ns_st_pu`	Publisher Brand Name	The consumer-facing brand name of the media publisher that owns the content	String	mandatory	▪ `CNN` ▪ `TBS` ▪ `EntertainmentWeekly` ▪ `CNET`

[3] Content segments of the same content asset should share the same content ID value.

[4] This is 24 minutes in milliseconds.

Attribute	Name	Description	Format	Mandatory?	Example
`ns_st_pr`	Program Name	The name of the overall program, show, or content series	String	mandatory if relevant and available	▪ `Survivor` ▪ `TheOffice` ▪ `CNET_Reviews` ▪ `HuffPost_Travel`
`ns_st_ep`	Episode Title or identification	The title of the specific episode or the season+episode number or a concatenation of both	String	mandatory if relevant and available	▪ `The_Hunt_Continues` ▪ `S02E14` ▪ `The_Hunt_Continues;S02E14`
`ns_st_cu`	Clip URL	URL of the content asset. Use value `none` if you cannot access the URL of the content asset.	String	mandatory	`http://video.server/asset/137849`
`ns_st_ad`	Advertisement Flag	Identifies the clip as an advertisement. If the attribute **is present** it means the flag is set.	Boolean	mandatory **only** for advertisement clips*(5)*	`1`
`ns_st_ct`	Classification Type	4-character ID which distinguishes advertisement stream types from content stream types	String	mandatory	`vc12`
`vmx_c3`	Video Metrix dictionary classification value #1	This value should be used if you have been tagging with the *legacy* "comScore Video Metrix Tag". Please pass the same value as you do in the `c3` parameter of the legacy Video Metrix Tag.	String	optional	

(5) Attribute `ns_st_ad` should **not** be used with content assets.

Attribute	Name	Description	Format	Mandatory?	Example
`vmx_c4`	Video Metrix dictionary classification value #2	This value should be used if you have been tagging with the *legacy* "comScore Video Metrix Tag". Please pass the same value as you do in the `c4` parameter of the legacy Video Metrix Tag.	String	optional	
`vmx_c6`	Video Metrix dictionary classification value #3	This value should be used if you have been tagging with the *legacy* "comScore Video Metrix Tag". Please pass the same value as you do in the `c6` parameter of the legacy Video Metrix Tag.	String	optional	

Detailed Description of Mandatory Attributes

This section contains detailed descriptions of the mandatory attributes mentioned in the previous table.

Important note: oftentimes, the values you will want to pass in these attributes will contain characters that cannot be passed in a URL (spaces, exclamation marks, commas, etc.). You do not have to URL encode or escape these characters. The Streaming Tag will take care of URL encoding these characters.

Clip Number

Attribute name: `ns_st_cn`

The clip number is used to identify the clip's position in the playlist. Clip numbers start counting from `1` – for the first clip in the playlist – and increase by one for each next clip. The clip number should increment for both content and ads.

Content ID

Attribute name: `ns_st_ci`

Please provide your internal unique content asset identifier so our report calculations logic can identify unique content assets. If your media player environment does not use or have access to a content asset identification please specify value `0` for the content ID.

For segmented content, `ns_st_ci` should share the same value for each content segment. For ads the value of `ns_st_ci` should be the content IDs of the ads. If the content IDs of the ads are not available then please reuse the content ID of the content segments for the ads.

Part Number

Attribute name: `ns_st_pn`

The part number is used to identify segments of content that are separated by mid-roll ads. Part numbers start counting from `1` – for the first segment in the playlist – and increase by 1 each time content is played after a mid-roll ad.

Total Number of Parts

Attribute name: `ns_st_tp`

The total number of parts is used to identify segments of content that are separated by mid-roll ads. The total number of parts is the number of content parts. In our example context on page 6 the content is split into three parts by mid-rolls and the value of `ns_st_tp` should be `3`.

When you do not know the total number of parts please set `ns_st_tp` to value `0`.

Clip Length

Attribute name: `ns_st_cl`

Please provide the length of the content or ad as the value of attribute `ns_st_cl`. These clip lengths need to be **in milliseconds**. Most media players will be able to provide these values as content metadata. In our example context on page 6 the value of attribute `ns_st_cl` for the pre-roll advertisement – clip number 1 – should be `20000`. The value of attribute `ns_st_cl` for the content – clip number 2 – should be `1440000`[6].

If the length of the content or ad is unknown or cannot be provided then please provide value `0`.

Publisher Brand Name

Attribute name: `ns_st_pu`

This parameter is used to provide detailed viewing/listening data for each of the individual, publisher brands your company owns. If you are a large media publisher, you may have many of these in your portfolio, both television/radio-native brands (e.g. television channels) as well as digital-native brands (e.g. websites). The name for the publisher brand should be passed in this parameter as you would like it to appear in comScore's public reports.

---

[6] This is calculated as (24 * 60 * 1000). Usually values provided by media players are not exact. It is not a problem if the value provided by the media player is not exactly the value shown in the player's user interface. If your player reports clip length as a number of seconds then please multiply any values by 1000 as needed.

Program Name

Attribute name: `ns_st_pr`

This parameter is used to provide granular viewing/listening data for each of the individual shows, programs, or content series that belong to the publisher brand listed in the `ns_st_pu` parameter. These may be television show names, radio show names, digital video/audio content series, or aggregations of certain genres of content owned by a publisher brand. The name for the show should be passed in this parameter as you would like it to appear in comScore's public reports.

Episode Title or Identification

Attribute name: `ns_st_ep`

This parameter is used to provide granular viewing/listening data for each of the individual episodes that belong to the program listed in the `ns_st_pr` parameter. "Episodes" may be traditional television/radio episodes or specific videos/audio-clips within a digital video/audio content series. The value for this parameter should be the consumer-facing title, the season+episode value (S##E##), or simply a numerical ID.

Clip URL

Attribute name: `ns_st_cu`

Please provide the URL of the streaming asset. This can be the URL of a stream or media for progressive download. Note: This should **not** be the URL of the page asset.

If you are no able to get the URL of the streaming asset in the implementation, please provide value `none`.

Classification Type

Attribute name: `ns_st_ct`

The `ns_st_ct` parameter is critical for enabling comScore to distinguish ad streams from content streams. Please use the tables below as a guide for determining the `ns_st_ct` value to place in the tag.

Video Streams

VIDEO Classification Types			`ns_st_ct` **value**
CONTENT*	**PREMIUM** Content with strong brand equity or brand recognition. Premium content is usually created or produced by media and entertainment companies using professional-grade equipment, talent, and production crews that hold or maintain the rights for distribution and syndication.	**Short Form** Video On Demand**	`vc11`
		Long Form Video On Demand**	`vc12`
		Live Streaming	`vc13`

VIDEO Classification Types			ns_st_ct value
	USER-GENERATED Content with little-to-no brand equity or brand recognition. User-generated content (UGC) has minimal production value, and is uploaded to the Internet by non-media professionals.	**Short Form** Video On Demand**	`vc21`
		Long Form Video On Demand**	`vc22`
		Live Streaming	`vc23`
	OTHER Used if none of the above categories apply.		`vc00`
AD*	**LINEAR – VIDEO ON DEMAND** Linear ads delivered into a media player and presented before, in the middle of, or after video content is consumed by the user. The ad completely takes over the full view of the media player.	**Linear Pre-Roll**	`va11`
		Linear Mid-Roll	`va12`
		Linear Post-Roll	`va13`
	LINEAR – LIVE Linear ads delivered before, in the middle of, or after a live stream of content. The ad completely takes over the full view of the media player.		`va21`
	OTHER Used if none of the above categories apply.		`va00`

* In cases where defining a video as ad or content is ambiguous, videos should be classified as content if they are monetizable. A video is monetizable if it could (or did) have video ads run against it. Conversely, a video should be classified as an ad if it is not in a position to have ads run against it due to the promotional nature of its subject matter.

** Long form video on demand is differentiated from short form video on demand in that long form content always has a content arc with a beginning, middle, and end which in its entirety typically lasts longer than 10 minutes.

IMPORTANT NOTE: The following types of video streams should NOT be tagged using the Streaming Tag unless otherwise directed by your comScore account team.

- **In-banner video ads**: In-banner video ads are the same as standard image/flv banner ads prevalent on the Internet, except they have a streamed video associated within them, (or consist entirely of a video). They leverage the banner space to deliver a video experience as opposed to another static or rich media format. The format relies on the existence of display ad inventory on the page for its delivery. Video banner ads can also have interactive rich media elements within them and can pop out of their banners to display larger video ads.
- **Overlay ads**: Overlay ads are non-linear video ads that are delivered as text, graphical banners/buttons, or as video and are placed within the media player window, either over the video content itself or directly on the top edge or bottom edge of the video content during the content play.
- **In-Text video ads**: In-text video ads are delivered as a pop over when a user chooses to mouse-over relevant, apparently hyperlinked words within a block of text.

- **Billboards or slates**: Billboards (also known as “slates”) are static promotional images usually run before video content and usually lasting fewer than 5 seconds with or without a voiceover. They are frequently not true video streams in the technical sense.

Audio Streams

AUDIO Classification Types	`ns_st_ct` value
CONTENT*	`ac00`
AD*	`aa00`

* In cases where defining an audio stream as ad or content is ambiguous, streams should be classified as content if they are monetizable. A stream is monetizable if it could (or did) have audio ads run against it. Conversely, an audio stream should be classified as an ad if it is not in a position to have ads run against it due to the promotional nature of its subject matter.

Video Metrix Dictionary Classification Values

Attribute names: `vmx_c3`, `vmx_c4`, `vmx_c6`

These values can be used if you have been tagging with the *legacy* “comScore Video Metrix Tag”. You do not have to use these attributes **unless** you have values which you were previously passing in the `c3`, `c4`, and `c6` parameters of the legacy “comScore Video Metrix Tag” which are not already provided in any other attributes.

For the values that are **not already provided** you can add the corresponding `vmx_c3`, `vmx_c4` or `vmx_c6` attributes. For values that are already present please contact your comScore account team to inform them in which attributes these values can be found.

For example:

- The value you were previously passing in the `c3` parameter is not present in any other attribute. Add the `vmx_c3` attribute with the value you were previously passing in the `c3` parameter.
- The value you were previously passing in the `c4` parameter is already present in attribute `ns_st_pu`. Inform your comScore account team they can find the value of the `c4` parameter in the `ns_st_pu` attribute.
- You were previously not using the `c6` parameter. Do not add the `vmx_c6` attribute.

If you are not familiar with what values you are currently passing in the legacy Video Metrix Tag, please contact your comScore account team.

Important note: oftentimes, the values you will want to pass in these parameters will contain characters that cannot be passed in a URL (spaces, exclamation marks, commas, etc.). The Streaming Tag will take care of URL encoding these characters.

Web Page URL, Web Page Title and Referring Web Page URL

In web browser environments the Streaming Tag will also try to communicate with the HTML DOM to automatically retrieve the web page URL, web page title and the referring web page URL (if available) of the HTML document where it is included. If these values can be retrieved then they will be added to the collected data.

Implementation Instructions

Implementation Checklist

Before deploying the plugin, please complete the following checklist first:

- Make sure you are using a media player that has a JavaScript API which allows you to detect the player state and get content details like the current playback position and content titles.
- Make sure you have obtained your comScore client ID. The client ID differentiates clients from one another and is provided to you by your comScore account team. You can also lookup your Client ID in the comScore Direct interface (see inset below).
- Make sure you have the Streaming Tag JavaScript library. Your comScore account team will have provided you with the library together with this document.
- Determine the values that need to be sent along with the collected data. See Collected Data on page 9 for more information about collected data.

Your comScore Client ID can be found in the comScore Direct interface.

1. Use a web browser to go to http://direct.comscore.com/clients/Video.aspx.
2. Log in to comScore Direct with your client login ID and password if you are prompted to. If you do not have a comScore login and already are a comScore client then please contact your client account team.
3. Confirm you are on the "Mobile App" tab and click on "Get Tag".
4. The Client ID – or, C2 value – is shown in the popup.

Include the Library

To start using the Streaming Tag you will have to make sure the JavaScript file with the Streaming Tag library is available in your HTML document. You can load the JavaScript from the HTML document's `<head>` section, using:

```
<script src="streamsense.min.js" type="text/javascript"></script>
```

This file contains JavaScript code with definitions of classes used by the Streaming Tag. You are allowed to load the file dynamically (or asynchronously) as long as you make sure the file is loaded before any of the code statements that use the Streaming Tag library is executed.

Initialization

To initialize the Stream Sense Tag, you need to create a new object instance, initialize it with a base measurement URL and store its reference:

```
var streamSense = new ns_.StreamSense({}, 'http://b.scorecardresearch.com/p?c1=2&c2=1234567');
```

See Base Measurement URL on page 8 for more information about the base measurement URL.

Set the Current Playlist

When your media player is going to play a collections of content assets and related advertisements you need to inform the Streaming Tag about a new playlist. You can call the `setPlaylist` API method to instruct the Streaming Tag to start a new playlist:

```
streamSense.setPlaylist();
```

When your media player is going to play a different content asset or ads that are related to a different content asset this usually indicates you should start a new playlist.

To make sure the Streaming Tag is aware there is a new playlist please make sure you only set the current playlist when the player is in a *stopped* state, i.e., after notifying of an *end* and before notifying of a *play*. These *state changes* are discussed in Notifying of Player State Changes on page 19.

Set the Current Clip

When the player loads new content or advertisements the current clip in the playlist should be identified. A clip is represented as a collection of attributes.

To set the current clip in the playlist you can call the `setClip` API method. This method accepts an object as a parameter to identify the current clip. This object should contain the attributes and values that apply to the current clip. Please refer to Collected Data on page 9 for more information about which attributes are available for use.

For example:
The media player is going to play the first segment of content as described in our example context on page 6. This is a long-form content stream which is assigned to clip 2 in the playlist. The stream is long-form content. The asset has the internal content ID “6352-2fe83a”. The content stream has a length of 24 minutes (1440000 milliseconds). The dictionary entity value is “MyChannel”. The playlist is maintained in array `clips` which contains the objects with clip attributes. Currently, `clips` only contains the attributes for clip number 1.

The second element of array `clips` – on index 1 – will correspond with clip number 2. The following code would set all the attributes using the values mentioned in this example in an object and add that to the `clips` array:

```
clips[1] = {
  ns_st_cn : "2",            // The clip number
  ns_st_ci : "6352-2fe83a",  // Internal Content ID
  ns_st_pn : "1",            // This is part (segment) 1 ...
  ns_st_tp : "3",            // ... of 3 parts in total
  ns_st_cu : "http://video.server/asset/13784",  // The clip URL
  ns_st_cl : "1440000",      // Length of the stream (milliseconds)
  ns_st_ct : "vc12"          // Classification Type
}
```

The `setClip` API method call to set the current clip to the second clip (on array index 1) would look like this:

```
streamSense.setClip(clips[1]);
```

To make sure the Streaming Tag is aware of the current clip and uses the correct details the use of the `setClip` API method is only allowed when the player is in a *stopped* state, i.e., **after** notifying of an *end* and **before** notifying of a *play*. These *state changes* are discussed in Notifying of Player State Changes on this page.

If your player supports mid-roll ads then please set the value of the part number attribute, `ns_st_pn`, accordingly before setting the current clip with your content details.

For example:
After playing the first mid-roll advertisement – i.e., clip 3 – in our example context the media player will continue with playing the second segment of the content asset. The content asset is the second clip in the playlist and its details were stored in the object in the second element of the `clips` array, as per the previous example.

The player is going to play the second segment of the content so the value of `ns_st_pn` on the second clip (on array index 1) needs to change to `2`, **before** setting the current clip:

```
clips[1].ns_st_pn = "2";
streamSense.setClip(clips[1]);
```

Notifying of Player State Changes

During use of the media player its state will change. To indicate player state changes to the Streaming Tag you can use the `notify` API method.

The `notify` API method takes three parameters:

state-change

Please specify a value from class `ns_.StreamSense.PlayerEvents` for this parameter. The table below specified which values are available.

transition attributes

This parameter is not used in a VideoMetrix implementation. Please always provide an empty object – i.e., `{}` – for this parameter.

current position

This parameter indicates the playback position – in **milliseconds** – at which the state transition took place. Please note that this value is relative to the **current** clip.

The second parameter, *state change specific attributes*, is not used in implementations for VideoMetrix reporting. Please always specify an object without any properties as the second parameter – i.e., by using `{}`.

The table below shows when the Streaming Tag should be notified of a particular state change:

Transition	Notified state-change
Media player starts buffering of the current clip (*this includes buffering during playback*)	`ns_.StreamSense.PlayerEvents.BUFFER`
Media player pauses playback of the current clip (*this also applies to seeking during playback*)	`ns_.StreamSense.PlayerEvents.PAUSE`
Media player starts or continues playback of the current clip (*this also applies to resuming playback after seeking*)	`ns_.StreamSense.PlayerEvents.PLAY`
Media player ends playback of the current clip	`ns_.StreamSense.PlayerEvents.END`

You can use this table to translate any internal player state change notifications into the correct call to the `notify` API method of the Streaming Tag API. You might notice other state changes identified in `ns_.StreamSense.PlayerEvents`. These are to support other implementation scenarios and should not be used.

To notify the Streaming Tag of a player state transition – e.g., `ns_.StreamSense.PlayerEvents.PLAY` – call the `notify` API method as follows:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.PLAY, {}, position);
```

As indicated above:

- the second parameter should **always** be an object without any properties
- `position` **must** be a value in milliseconds

Auto-repeat (looping)

Some media players will automatically repeat playback when they naturally reach the end of the media they were playing. In the Streaming Tag implementation model this corresponds with *looped* playback of the entire playlist.

The Streaming Tag should be notified of this *auto-repeat* behavior as follows:

1. When the player reaches the end of the content, notify the Streaming Tag of the end of playback.
2. Instruct the Streaming Tag to set the current clip. The current clip should correspond with the content that will be played when the media player automatically repeats playback. In most cases this will be the first clip of the playlist. When you set the current clip, please specify `true` for the second parameter which identifies looped playback (see the example below).
3. When the player starts repeated playback, notify the Streaming Tag of the start of playback (or buffering, if applicable).

Please identify playback has looped by providing value `true` as the second parameter of the `setClip` method call when you tag auto-repeat behavior.

Please remember to specify the correct part number for the current clip for cases where mid-roll ads were played. When the content will start repeated playback from the beginning after looping then the part number – i.e., attribute `ns_st_pn` – should be set to value `1`.

For example:
The media player is playing the post-roll ad from our example context on page 6. The ad reaches the end and the media player will automatically repeat playback. It will skip the pre-roll, starting from the first segment of content which is the second clip of the playlist. The playlist is maintained in array `clips` which contains the objects with clip attributes.

In this example the part number of the content clip – at index 1 in the `clips` array – will have to be set to value `1` after ending playback of the post-roll ad. The Streaming Tag API method calls would look like:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.END, {}, position);
clips[1].ns_st_pn = "1";
streamSense.setClip(clips[1], true);
streamSense.notify(ns_.StreamSense.PlayerEvents.PLAY, {}, position);
```

Scrubbing and/or Seeking

When the user starts scrubbing or seeking **during playback**, notify the Streaming Tag of a transition to **paused** state:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.PAUSE, {}, position);
```

The value of `position` should reflect the *source* position (in milliseconds) – i.e., where seeking **started**.

When the user ends scrubbing or seeking and playback continues, notify the Streaming Tag of a transition to **resumed** state:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.PLAY, {}, position);
```

The value of `position` should reflect the *target* position (in milliseconds) – i.e., where seeking **ended**.

The Streaming Tag only needs to be notified of seeking if seeking occurs **during playback**. If the user seeks while the player is paused then notifications are not required[7].

For example:
The user seeks from position 3 minutes and 25 seconds to position 10 minutes and 43 seconds.

[7] The change in position will be measured when playback resumes.

Start of seeking should be tagged with a call to the `notify` API method, using `ns_.StreamSense.PlayerEvents.PAUSE` with position `205000`.
End of seeking should be tagged with a call to the `notify` API method, using `ns_.StreamSense.PlayerEvents.PLAY` with position `643000`.

Buffering During Playback

If the player starts buffering content during starting playback then notify the Streaming Tag of buffering.

```
streamSense.notify(ns_.StreamSense.PlayerEvents.BUFFER, {}, position);
```

You do not have to notify the Streaming Tag of any state changes as long as the player continues to buffer. When the player finishes buffering and playback resumes you can notify the Streaming Tag of playback:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.PLAY, {}, position);
```

If the player cannot recover from buffering and playback ends then please notify the Streaming Tag playback has ended:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.END, {}, position);
```

Pre-buffering Without Starting Playback

If the player is capable of buffering content without immediately starting playback – i.e., pre-buffering or auto-buffering – then it is allowed to notify the Streaming Tag of buffering when the player starts pre-buffering:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.BUFFER, {}, position);
```

When pre-buffering is completed, notify the Streaming Tag of a transition to **paused** state:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.PAUSE, {}, position);
```

At this point the player might remain idle until the user instructs the player to start playing. When playback starts, the Streaming Tag can be notified of the start of playback:

```
streamSense.notify(ns_.StreamSense.PlayerEvents.PLAY, {}, position);
```

In all three statements the value of `position` should reflect the position where the player will start playing the content (**in milliseconds**) or value `0` if the position is unknown or cannot be determined at this time.

Testing the Implementation

To verify the implementation you can load a web page which contains your player and Streaming Tag implementation in a web browser. Please use an HTTP sniffer to review the HTTP traffic.

When your player starts playing content you should see HTTP requests to host *b.scorecardresearch.com* which contain your comScore client ID in the `c2` query string parameter together with the collected data. The measurement data should match with your clip attributes and should be sent on player state transitions.

If you have completed the implementation then please verify the implementation and collected data with your Client Service Representative.