



Adobe® Marketing Cloud
Video Analytics Implementation Guide 1.5 for
JavaScript

Contents

Video Analytics Implementation Guide 1.5 for JavaScript.....	4
Getting started on JavaScript.....	4
Download the SDK.....	4
Implement the JavaScript library.....	5
Implementation Guide.....	6
Configure AppMeasurement.....	6
How the JavaScript VideoPlayerPluginDelegate Works.....	7
Implement VideoPlayerPluginDelegate - JS.....	8
Attaching Custom Metadata.....	11
Standard Metadata Parameters.....	12
Standard metadata keys for JavaScript.....	14
Sample implementation on JavaScript.....	15
Configure the Video Heartbeat Library.....	16
Track Player Events.....	17
Track Methods and Player Events.....	19
Test Your Video Measurement Code.....	21
Video Measurement Parameters.....	22
Sample player.....	26
Debugging.....	26
Enable Debug Logging.....	26
Validate implementations.....	27
Adobe Debug.....	27
Heartbeats parameters.....	28
Adobe Analytics parameters.....	32
Ratings Partners Integration.....	33
Scenarios.....	33
Scenario and Timeline Illustrations.....	33
Tracking Explained.....	35
Non-Linear Tracking Scenarios.....	51
Pause tracking.....	52

Contact and Legal Information.....55

Video Analytics Implementation Guide 1.5 for JavaScript

This section contains instructions to download the video heartbeat SDKs and developer guides for your platform. Make sure you also download the developer guide that is in the *docs* folder when you download the SDK as it contains the specific implementation instructions for video heartbeat.

Platform	Process
JavaScript	See Implementation Guide .

Getting started on JavaScript

Before you can use Video Heartbeat 1.5x and 1.6.x in JavaScript, you must complete a few tasks.

Setting up the Marketing Cloud account

To set up the Marketing Cloud account, contact an Adobe representative. After the Marketing Cloud account is set for video analytics, you must enable the Visitor ID service to use Video Heartbeat.

Prerequisites to implementing

Before you start implementing Video Heartbeat for Android in the next section, ensure that you have completed the following tasks:

- Valid implementation for ADBMobile for JavaScript in your application.

For more information about the Adobe Mobile SDK documentation, see [Android SDK 4.x for Marketing Cloud Solutions](#).

- Visitor ID service should be implemented.

For more information about the Visitor ID service, see [Marketing Cloud ID Service](#).

- Valid configuration parameters for Video Heartbeat.

These parameters can be obtained from an Adobe representative after you set up the video analytics account.

- This guide is intended for a media integration engineer who has an understanding of the APIs and workflow of the media player being instrumented. Implementing these APIs requires that your media player provide the following:

- An API to subscribe to player events.

The media heartbeat requires that you call a set of simple APIs when events occur in your player.

- An API or class that provides player information, such as the media name and play head position.

Download the SDK

The video heartbeat library is distributed using a public Github repository.

1. Browse to [Adobe Github Video Heartbeat](#) and download the latest release for your platform.
2. Extract the zip, and copy `VideoHeartbeat.min.js` to a location accessible to your project. Optionally, copy the non-minified version to your project for debugging.
3. Save the `samples` folder to a location where the sample project can be reviewed and tested.

Your next step is to [Configure AppMeasurement](#).

Implement the JavaScript library

After you download the Android SDK and add it to your project, you can collect video metrics, such as initiates, content starts, ad starts, ad completes, content completes and so on.

Get the JavaScript SDK

Before you get the SDK, you must set up a mobile SDK and download the Video Heartbeat SDK. For more information, see [Getting started on JavaScript](#).

1. Expand the `VideoHeartbeatLibrary-android-v2.*.zip` file that you downloaded.

For more information about downloading this file, see [Getting started on JavaScript](#).

2. Verify that the `VideoHeartbeat.jar` file exists in the `libs` directory:

This library is used with Android devices and simulators for video heartbeat tracking APIs.

Add the SDK to your project

To add the SDK to your IntelliJ IDEA project:

1. Right click your project in the **Project navigation** panel.
2. Select **Open Module Settings**.
3. Under **Project Settings**, select **Libraries**.
4. Click **+** to add a new library.
5. Select **Java** and navigate to the `VideoHeartbeat.jar` file.
6. Select the modules where you plan to use the mobile library.
7. Click **Apply** and then **OK** to close the Module Settings window.

To add the SDK to your Eclipse project:

1. In the Eclipse IDE, right-click on the project name.
2. Click **Build Path > Add External Archives**.
3. Select `VideoHeartbeat.jar`.
4. Click **Open**.
5. Right-click the project again, and click **Build Path > Configure Build Path**.
6. Click the **Order** and **Export** tabs.
7. Ensure that the `VideoHeartbeat.jar` file is selected.

Adding app permissions

The VideoHeartbeat Library requires the following permissions to send data in tracking calls:

- `INTERNET`
- `ACCESS_NETWORK_STATE`

To add these permissions, add the following lines to your `AndroidManifest.xml` file in the application project directory:

- `<uses-permission android:name="android.permission.INTERNET" />`
- `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />`

Implementation Guide

This guide describes how to add video heartbeat measurement to any video player that provides a JavaScript API. Most web-based players provide a JavaScript API, even if the underlying video is delivered in Flash or other video formats.

This section was last updated 06/18/2015.

For example, the following players can be tracked using JavaScript:

- YouTube
- Brightcove
- Kaltura
- Ooyala
- HTML 5 (using native JavaScript support in the web browser and the [HTML 5 video events](#)).

For other players, implementing video heartbeat measurement requires that your video player provides a JavaScript API with the following:

- An API to subscribe to player events. The video heartbeat SDK requires that you call a set of simple functions as actions occur in your player.
- An API or class that provides player information, such as video name and playhead location. The video heartbeat SDK requires that you implement an interface that returns current video information.

Requirements

Integrating video heartbeat library requires the following:

- Existing Analytics implementation. These instructions assume that you have an existing implementation of AppMeasurement that is also using the Marketing Cloud Visitor ID Service. To implement Analytics or the Marketing Cloud Visitor ID Service, see the [Adobe Analytics Implementation Guide](#) and the [Marketing Cloud Visitor ID Service Guide](#).
- Video heartbeat library. (download instructions are in this guide)



Note: Make sure your Analytics implementation is configured to send data to a development report suite before you start development.

Example Implementations

An example is available in the `samples` folder that is included with the video heartbeat library.

Implementation Process

Complete the following steps to add video heartbeat tracking to your player:

Configure AppMeasurement

The JavaScript implementation is configured similar to the ActionScript implementation on your website. The standard [Analytics Variables](#) are all available. Video Heartbeat also requires that you implement the [Marketing Cloud visitor ID service](#).

1. Instantiate and configure the Marketing Cloud visitor ID service:

```
// Visitor
var visitor = new Visitor("INSERT-MCORG-ID-HERE");
visitor.trackingServer = "INSERT-TRACKING-SERVER-HERE";
```

2. Instantiate and configure AppMeasurement:

```
// AppMeasurement
var appMeasurement = new AppMeasurement();
appMeasurement.visitor = visitor;
appMeasurement.trackingServer = <tracking-server>;
appMeasurement.account = <rsid>;
// ... other AppMeasurement-specific configs (e.g., pageName, currency etc.)
```

At a minimum, configure the following three variables:

- `appMeasurement.account`
- `appMeasurement.trackingServer`
- `appMeasurement.visitor`

Next step: [Implement VideoPlayerPluginDelegate - JS](#).

How the JavaScript VideoPlayerPluginDelegate Works

Examples to understand the interaction between the player event listeners, the track functions, and the VideoPlayerPluginDelegate on JavaScript.



Note: This video player plugin delegate was previously named `PlayerDelegate` in version 1.4.

If you have reviewed the [Track Methods and Player Events](#) topic, you might have noticed that none of the track methods take any parameters. Instead of passing video name, playhead information, and chapter information directly to these methods, video heartbeat uses a `VideoPlayerPluginDelegate` class (`ADB_VHB_VideoPlayerPluginDelegate` on iOS) that is queried for this information instead. As part of your implementation, you are required to extend this class to provide specific information about your player.

To understand the interaction between the player event listeners, the track functions, and the `VideoPlayerPluginDelegate`, consider the following examples:

Event Listeners

VideoPlayerPlugin Track Functions

In the `trackVideoPlay()` JavaScript function you assigned to handle the `play` event, you would call `VideoPlayerPlugin.trackPlay()` to let video heartbeat know that playback has started:

```
function trackVideoPlay() {
    VideoPlayerPlugin.trackPlay();
};
```

Note that no video information is passed to the `trackPlay()`.

VideoPlayerPluginDelegate

When the video heartbeat `track...` methods are called, your implementation of `VideoPlayerPluginDelegate` is queried automatically as needed to provide any required details about the video, ad, or chapter. This removes the need for you to

determine exactly what information is needed by each track function. You can provide a single object that returns the most current information available. The following is a simple example:

```
function SampleVideoPlayerPluginDelegate(player) {
    this._player = player;
}

SampleVideoPlayerPluginDelegate.prototype.getVideoInfo = function() {
    var videoInfo = new VideoInfo();
    videoInfo.id = this._player.getVideoId(); // e.g. "vid123-a"
    videoInfo.name = this._player.getVideoName(); // e.g. "My sample video"
    videoInfo.length = this._player.getVideoLength(); // e.g. 240 seconds
    videoInfo.streamType = AssetType.ASSET_TYPE_VOD;
    videoInfo.playerName = this._player.getName(); // e.g. "Sample video player"
    videoInfo.playhead = this._player.getCurrentPlayhead(); // e.g. 115 (obtained from the
video player)
    return videoInfo;
};

SampleVideoPlayerPluginDelegate.prototype.getAdBreakInfo = function() {
    return null; // no ads in this scenario
};

SampleVideoPlayerPluginDelegate.prototype.getAdInfo = function() {
    return null; // no ads in this scenario
};

SampleVideoPlayerPluginDelegate.prototype.getChapterInfo = function() {
    return null; // no chapters in this scenario
};

SampleVideoPlayerPluginDelegate.prototype.getQoSInfo = function() {
    return null; // no QoS information in this sample
};
```



Note: The `onError` callback that was part of the `PlayerDelegate` in version 1.4 is removed from the `VideoPlayerPluginDelegate` in version 1.5.

In this example, when `trackPlay` is called, your instance of `VideoInfo` is read to determine the current offset of the video to calculate time played. The querying happens automatically: you are required only to extend `VideoPlayerPluginDelegate` and provide an instance of the extended class as a parameter to `VideoPlayerPlugin` when you initialize video heartbeat.

Make sure you take a close look at the sample players to see how `VideoPlayerPluginDelegate` is extended.

Implement VideoPlayerPluginDelegate - JS

The `VideoPlayerPluginDelegate` is used by video heartbeat to get information about the currently playing video, ad, and chapter.



Note: This video player plugin delegate was previously named `PlayerDelegate` in version 1.4.

The `VideoPlayerPluginDelegate` interface is where you will typically spend a majority of your implementation time.

To get started creating your own `VideoPlayerPluginDelegate` implementation, create a new object that uses `ADB.va.VideoPlayerPluginDelegate` as the object prototype:

```
var myDelegate = new VideoPlayerPluginDelegate();
```

Now that you have a player delegate, you need to define the functions that return information about your video and player:

```
function VideoPlayerPluginDelegate() {}

VideoPlayerPluginDelegate.prototype.getVideoInfo = function() {};

VideoPlayerPluginDelegate.prototype.getAdBreakInfo = function() {};
```



```

VideoPlayerPluginDelegate.prototype.getAdInfo = function() {};
VideoPlayerPluginDelegate.prototype.getChapterInfo = function() {};
VideoPlayerPluginDelegate.prototype.getQoSInfo = function() {};

```

With that framework in place, the following sections explain how to update these methods to return useful data from your player:

- [Video Information](#)
- [Ad Break Information](#)
- [Ad Information](#)
- [Chapter Information](#)
- [Example](#)

Video Information

The `getVideoInfo` method returns a `VideoInfo` object that contains details about the video player and the currently playing video. Before you can define this object, you'll need to use the API documentation provided by your player to find out how video information is retrieved. Video information is usually a property of the player object or retrieved using a private method.

For example, In HTML 5, the playhead is a property of the `<video>` element:

```
document.getElementById('movie').currentTime;
```

In the YouTube API, the playhead is returned by a method call exposed by the player:

```
player.getCurrentTime();
```

To implement your custom `getVideoInfo` method, you'll need the following information:

Parameter	Required?	Description
<code>playerName</code>	Yes	The name of the video player that is playing back the main content.
<code>id</code>	Yes	The ID of the video asset.
<code>name</code>	No	The name of the video asset (opaque string value).
<code>length</code>	Yes	The duration (in seconds) of the video asset. If <code>streamType</code> is set to <code>vod</code> , return the length of the video. For other video types, return -1 as the length.
<code>playhead</code>	Yes	The current playhead location (in seconds) inside the video asset (excluding ad content) at the moment this method was called.
<code>streamType</code>	Yes	The type of the video asset.
<code>resumed</code>	No	Set to <code>true</code> if this is a resumed video playback session (for example, when playback of VOD content starts from where the user previously left it).

After you have figured out how to get the required information, update the `getVideoInfo` method to return a `VideoInfo` object with the video information. How you populate each value is up to you, and varies based on your player. For example, you might load the video player name using a configuration file, or you could hard-code the value if you use only one player.

Ad Break Information

Ad breaks provide insight as to when a particular ad was displayed. For example, if you have a pre-roll and a midpoint ad break, you can collect position data along with the specific ad data. If you have only one ad break, you can simply provide 1 for the position and leave the name blank.

Parameter	Required?	Description
playerName	Yes	The name of the video player responsible with playing back the current advertisement break.
name	No	The name of the ad-break.
position	Yes	The position (index) of the pod inside the main content (starting with 1).
startTime	No	The offset of the ad-break inside the main content (in seconds). Defaults to the playhead inside the main content at the moment of the <code>trackAdStart</code> call.

Ad Information

Ad information is retrieved using a similar process used to retrieve video information, except you return an `AdInfo` object instead with details about the currently playing video ad. Use the API documentation provided by your Ad vendor to determine the following:

Parameter	Required?	Description
id	Yes	The ID of the ad asset.
length	Yes	The duration (in seconds) of the ad asset.
position	Yes	The position (index) of the ad inside the parent ad-break (starting with 1).
name	No	The name of the ad asset (opaque string value).

After you have figured out how to get the required information, update the `getAdInfo` method to return an `AdInfo` object with the ad information.

Chapter Information

If you are tracking chapters, you'll need to coordinate the chapter information returned with each call you make to `trackChapterStart`. Since chapters are likely defined by you and not your video player, you'll need a way to retrieve chapter definitions to populate this object.

Parameter	Required?	Description
name	No	The name of the chapter (opaque string value).
length	Yes	The duration (in seconds) of the chapter.
position	Yes	The position of the chapter inside the main content (starting from 1).
startTime	Yes	The offset inside the main content where the chapter starts.

Update the `getChapterInfo` method to retrieve properties or call the required APIs.

Example

The following is an example of a valid video player plugin delegate:

```
function SampleVideoPlayerPluginDelegate(player) {  
    this._player = player;  
}
```

```
SampleVideoPlayerPluginDelegate.prototype.getVideoInfo = function() {
    var videoInfo = new VideoInfo();
    videoInfo.id = this._player.getVideoId(); // e.g. "vid123-a"
    videoInfo.name = this._player.getVideoName(); // e.g. "My sample video"
    videoInfo.length = this._player.getVideoLength(); // e.g. 240 seconds
    videoInfo.streamType = AssetType.ASSET_TYPE_VOD;
    videoInfo.playerName = this._player.getName(); // e.g. "Sample video player"
    videoInfo.playhead = this._player.getCurrentPlayhead(); // e.g. 115 (obtained from the
video player)
    return videoInfo;
};

SampleVideoPlayerPluginDelegate.prototype.getAdBreakInfo = function() {
    return null; // no ads in this scenario
};

SampleVideoPlayerPluginDelegate.prototype.getAdInfo = function() {
    return null; // no ads in this scenario
};

SampleVideoPlayerPluginDelegate.prototype.getChapterInfo = function() {
    return null; // no chapters in this scenario
};

SampleVideoPlayerPluginDelegate.prototype.getQoSInfo = function() {
    return null; // no QoS information in this sample
};
```

Next step: [Configure the Video Heartbeat Library](#)

Attaching Custom Metadata

You can attach your own metadata to calls made to Adobe Analytics.

The video heartbeat library provides support for custom metadata to be attached to the analytics calls. The relevant APIs for this functionality are defined on the `AdobeAnalyticsPlugin`:

```
AdobeAnalyticsPlugin.prototype.setVideoMetadata = function(data) {};
AdobeAnalyticsPlugin.prototype.setAdMetadata = function(data) {};
AdobeAnalyticsPlugin.prototype.setChapterMetadata = function(data) {};
```

The integration code may call these methods on the `AdobeAnalyticsPlugin` to set custom metadata for the video, the ad and/or the chapter. The metadata for the video will be automatically associated with the ads and chapters.

You need to set the metadata before calling the relevant `track...()` method on the `VideoPlayerPlugin` by completing the following tasks:

- Set the video metadata before calling `trackVideoLoad()`
- Set the ad metadata before calling `trackAdStart()`
- Set the chapter metadata before calling `trackChapterStart()`

This will ensure that the metadata is taken into consideration by the video heartbeat library when processing the `track...()` call.

The code snippet below illustrates how to set custom metadata for video, ads and chapters:

```
// Before calling trackVideoLoad():
adobeAnalyticsPlugin.setVideoMetadata({
  isUserLoggedIn: "false",
  tvStation: "Sample TV station",
  programmer: "Sample programmer"
});

// [...]

// Before calling trackAdStart():
adobeAnalyticsPlugin.setAdMetadata({
  affiliate: "Sample affiliate",
  campaign: "Sample ad campaign"
});

// [...]

// Before calling trackChapterStart():
adobeAnalyticsPlugin.setChapterMetadata({
  segmentType: "Sample segment type"
});
```



Note: Clearing the custom metadata - The custom metadata set on the `AdobeAnalyticsPlugin` is persistent. It is not reset automatically by the video heartbeat library. To clear the custom metadata, you can pass `NULL` as the input argument for each of the `set...Metadata()` methods. For example, you should do this for ads and chapters once they are complete. Otherwise, the custom metadata will be applied to subsequent ads / chapters. It is your responsibility to ensure that the appropriate metadata is set before the `trackVideoLoad()` / `trackAdStart()` / `trackChapterStart()` call.

Standard Metadata Parameters

List of data-collection parameters sent by video heartbeat.



Important: This feature is valid only on Mobile and JavaScript platforms.

This section contains the following:

- [Video metadata](#)
- [Ad metadata](#)

Video metadata

Name	Context data key	Description	Metadata key name
Show	<code>a.media.show</code>	Data type: String	SHOW
Season	<code>a.media.season</code>	Data type: String	SEASON
Episode	<code>a.media.episode</code>	Data type: String	EPISODE
Asset ID	<code>a.media.asset</code>	This is the TMS_ID, an industry standard ID to identify a piece of TV/video content. TMS = Tribune Media Service, which is now known as Gracenote. Data type: String	ASSET_ID
Genre	<code>a.media.genre</code>	Data type: String	GENRE
First air date	<code>a.media.airDate</code>	Original TV air date of the asset. Data type: String	FIRST_AIR_DATE
First Digital Date	<code>a.media.digitalDate</code>	First date when this video asset was available on Digital. Data type: String	FIRST_DIGITAL_DATA
Content Rating	<code>a.media.rating</code>	Data type: String	RATING
Originator	<code>a.media.originator</code>	Data type: String	ORIGINATOR
Network	<code>a.media.network</code>	Data type: String	NETWORK
Show type	<code>a.media.type</code>	Data type: String	SHOW_TYPE
Ad Loads	<code>a.media.adLoad</code>	Data type: String	AD_LOAD
MVPD	<code>a.media.pass.mvpd</code>	Data type: String	MVPD
Authorized	<code>a.media.pass.auth</code>	Data type: String	AUTHORIZED
Day Part	<code>a.media.dayPart</code>	Data type: String	DAY_PART
Feed Type	<code>a.media.feed</code>	This determines the type of feed. For example, for living programming, the feed types are <i>East HD</i> or <i>West HD</i> .	FEED

Name	Context data key	Description	Metadata key name
		Data type: String	
Stream Format	a.media.format	Clip Classification. If the content is a full episode, pass a value of 1; otherwise pass a value of 0. The default value is 0. Data type: String	STREAM_FORMAT

Ad metadata

Property name	Context data key	Description	Metadata key name
Advertiser	a.media.ad.advertiser	The company or brand whose product is featured in the ad. Data Type: String	ADVERTISER
Campaign ID	a.media.ad.campaign	Client paramaters. Data Type: String	CAMPAIGN_ID
Creative ID	a.media.ad.creative	Client paramaters. Data Type: String	CREATIVE_ID
Placement ID	a.media.ad.placement	Client paramaters. Data Type: String	PLACEMENT_ID
Site ID	a.media.ad.site	Client paramaters. Data Type: String	SITE_ID
Creative URL	a.media.ad.creativeURL	The URL of the creative or ad that is being delivered. Data Type: String	CREATIVE_URL

Standard metadata keys for JavaScript

Here are the standard metadata keys for JavaScript:

```
ADB.va.plugins.aa.VideoMetadataKeys = {
  SHOW,
  SEASON,
  EPISODE,
  ASSET_ID,
  GENRE,
  FIRST_AIR_DATE,
  FIRST_DIGITAL_DATE,
  RATING,
```

```
ORIGINATOR,  
NETWORK,  
SHOW_TYPE,  
AD_LOAD,  
MVPD,  
AUTHORIZED,  
DAY_PART,  
FEED,  
STREAM_FORMAT
```

```
};
```

```
ADB.va.plugins.aa.AdMetadataKeys = {
```

```
ADVERTISER,  
CAMPAIGN_ID,  
CREATIVE_ID,  
PLACEMENT_ID,  
SITE_ID,  
CREATIVE_URL
```

```
};
```

Sample implementation on JavaScript

Here is a sample implementation on JavaScript.

To set standard metadata or partner metadata information, the application must use context metadata APIs and set the expected key-value pair by using one of the following APIs on `AdobeAnalyticsPlugin`:

- `setVideoMetadata` - for setting video metadata
- `setAdMetadata` - for setting ad metadata

To attach custom metadata and standard metadata keys, see the following information:

- [Attaching Custom Metadata](#)
- [Standard metadata keys for JavaScript](#)



Tip: The class that implements Video Analytics is `VideoAnalyticsProvider`.

```
// import Standard Metadata namespace  
var VideoMetadataKeys = ADB.va.plugins.aa.VideoMetadataKeys;  
var AdMetadataKeys = ADB.va.plugins.aa.AdMetadataKeys;  
  
// setting Standard Video Metadata  
var contextData = {};  
  
contextData[VideoMetadataKeys.SEASON] = "sample season";  
contextData[VideoMetadataKeys.SHOW] = "sample show";  
contextData[VideoMetadataKeys.EPISODE] = "sample episode";  
contextData[VideoMetadataKeys.ASSET_ID] = "sample asset id";  
contextData[VideoMetadataKeys.GENRE] = "sample genre";  
contextData[VideoMetadataKeys.FIRST_AIR_DATE] = "sample air date";  
contextData[VideoMetadataKeys.FIRST_DIGITAL_DATE] = "sample digital date";  
contextData[VideoMetadataKeys.RATING] = "sample rating";  
contextData[VideoMetadataKeys.ORIGINATOR] = "sample originator";  
contextData[VideoMetadataKeys.NETWORK] = "sample network";  
contextData[VideoMetadataKeys.SHOW_TYPE] = "sample show type";  
contextData[VideoMetadataKeys.AD_LOAD] = "sample ad load";  
contextData[VideoMetadataKeys.MVPD] = "sample mvpd";  
contextData[VideoMetadataKeys.AUTHORIZED] = "true";  
contextData[VideoMetadataKeys.DAY_PART] = "sample day part";  
contextData[VideoMetadataKeys.FEED] = "sample feed";  
contextData[VideoMetadataKeys.STREAM_FORMAT] = "sample format";  
  
// setting Standard Ad Metadata  
  
var contextData = {};  
  
contextData[AdMetadataKeys.ADVERTISER] = "sample advertiser";  
contextData[AdMetadataKeys.CAMPAIGN_ID] = "sample campaign";  
contextData[AdMetadataKeys.CREATIVE_ID] = "sample creative";  
contextData[AdMetadataKeys.CREATIVE_URL] = "sample url";  
contextData[AdMetadataKeys.SITE_ID] = "sample site";  
contextData[AdMetadataKeys.PLACEMENT_ID] = "sample placement";  
  
this._aaPlugin.setAdMetadata(contextData);
```

Configure the Video Heartbeat Library

You can configure each of the video heartbeat library components on an individual basis.

After you [Implement VideoPlayerPluginDelegate](#), you are ready to add the video heartbeat code to your project. Before you proceed, make sure you have the following:

- An instance of your custom `VideoPlayerPluginDelegate` object.
- A properly configured `ADBMobileConfig.json` file.

For more information, see [Configure AppMeasurement](#).

- Your Marketing Cloud Org ID or Publisher ID (assigned by Adobe).



Note: Existing customers using the Publisher ID can continue using it, but we recommend that you start using your Marketing Cloud Org ID instead. Contact Adobe Customer Care to obtain a Marketing Cloud Org ID.

On each HTML page where you are tracking video, add a `<script>` tag with a reference to `VideoHeartbeat.min.js`:

```
<script src="VideoHeartbeat.min.js"></script>
```

The following code sample illustrates how to instantiate and configure the video heartbeat components:

```
// Video Player plugin
var vpPluginDelegate = new CustomVideoPlayerPluginDelegate(<my-player>);
var vpPlugin = new VideoPlayerPlugin(vpPluginDelegate);
var vpPluginConfig = new VideoPlayerPluginConfig();
vpPluginConfig.debugLogging = true; // set this to false for production apps.
vpPlugin.configure(vpPluginConfig);

// Adobe Analytics plugin
var aaPluginDelegate = new CustomAdobeAnalyticsPluginDelegate();
var aaPlugin = new AdobeAnalyticsPlugin(appMeasurement, aaPluginDelegate);
var aaPluginConfig = new AdobeAnalyticsPluginConfig();
aaPluginConfig.channel = <syndication-channel>;

aaPluginConfig.debugLogging = true; // set this to false for production apps.
aaPlugin.configure(aaPluginConfig);

// Adobe Heartbeat plugin
var ahPluginDelegate = new CustomAdobeHeartbeatPluginDelegate();
var ahPlugin = new AdobeHeartbeatPlugin(ahPluginDelegate);
var ahPluginConfig = new AdobeHeartbeatPluginConfig(<tracking-server>, <publisher>);
ahPluginConfig.ovp = <online-video-platform-name>;
ahPluginConfig.sdk = <player-SDK-version>;
ahPluginConfig.debugLogging = true; // set this to false for production apps.
ahPluginConfig.ssl = false; // set this to true to enable Heartbeat calls through HTTPS
ahPlugin.configure(ahPluginConfig);

// Heartbeat
var plugins = [vpPlugin, aaPlugin, ahPlugin];
var heartbeatDelegate = new CustomHeartbeatDelegate();
var heartbeat = new Heartbeat(heartbeatDelegate, plugins);
var heartbeatConfig = new HeartbeatConfig();
heartbeatConfig.debugLogging = true; // set this to false for production apps.
heartbeat.configure(heartbeatConfig);
```

The configuration of each of the video heartbeat components follows the builder pattern:

- A configuration object is built
- The configuration object is passed as a parameter to the `configure` method of the component

The list below describes all the configuration parameters:

- **VideoPlayerPlugin**

- `debugLogging`: activates logging inside this plugin. Optional. Default value: **false**
- **AdobeAnalyticsPlugin**
 - `channel`: the name of the syndication channel. Optional. Default value: **the empty string**
 - `debugLogging`: activates logging inside this plugin. Optional. Default value: **false**
- **AdobeHeartbeatPlugin**
 - **trackingServer**: the server to which all the heartbeat calls are sent. Mandatory. Use the value provided by your Adobe consultant.
 - **publisher**: the name of the publisher. Mandatory. Use the value provided by your Adobe consultant.
 - `ssl`: Indicates whether the heartbeat calls should be made over HTTPS. Optional. Default value: **false**
 - `ovp`: the name of the online video platform through which content gets distributed. Optional. Default value: **"unknown"**
 - `sdk`: the version of the video player app/SDK. Optional. Default value: **"unknown"**
 - `quietMode`: activates the "quiet" mode of operation, in which all output HTTP calls are suppressed. Default value: **false**
 - `debugLogging`: activates logging inside this plugin. Optional. Default value: **false**
- **Heartbeat**
 - `debugLogging`: activates logging within the core Heartbeat component. Optional. Default value: **false**



Note: Setting the `debugLogging` flag to true on any of the video heartbeat components will activate fairly extensive tracing messaging which may impact performance. While these messages are useful during development and debugging, you should set all `debugLogging` flags to false for the production version of your player app. Note that the `debugLogging` flags default to false, so logging is disabled by default.

Test Your Configuration

Before you continue, load your code in a browser to make sure everything loads without errors. Optionally, set the `debugLogging` flag to true while you test:

```
heartbeatConfig.debugLogging = true; // remove or set to false for production!
```

Next, open your code in a browser and check the JavaScript console for errors (the code must be running on a local or remote web server). If there are no errors, you can use the JavaScript console to make a call to `trackVideoLoad()` and then `trackPlay()` to simulate a video play. If you check the network tab, you'll see a call to your Analytics data collection server, and additional calls to the Video Heartbeat tracking server.

After you test your configuration, continue to [Track Player Events](#).

Track Player Events

Media players that provide JavaScript event handlers are typically tracked by attaching callback functions to the video player event handlers.

The next step is to call the video heartbeat track methods when specific events occur in your player. This typically involves subscribing to events, registering a callback function, and then calling the correct method in the callback. Review the [Track Methods and Player Events](#) sections for details on exactly which method you should call for each corresponding player event.

The following example demonstrates event handling for HTML 5 video:

```
var myvideo = document.getElementById('movie');  
myvideo.addEventListener('play', trackPlay, false);
```

```
myvideo.addEventListener('ended', trackComplete, false);
myvideo.addEventListener('seeked', seekEnd, false);
myvideo.addEventListener('seeking', seekStart, false);
myvideo.addEventListener('pause', pause, false);
myvideo.addEventListener('ended', complete, false);

function trackPlay() {
  var myvideo = document.getElementById('movie');
  if (myvideo.currentTime == 0) {
    vpPlugin.trackVideoLoad();
    vpPlugin.trackPlay();
  } else {
    vpPlugin.trackPlay();
  }
}

function pause(e) {
  vpPlugin.trackPause();
}

function seekStart(e) {
  vpPlugin.trackSeekStart();
}

function seekEnd(e) {
  vpPlugin.trackSeekComplete();
}

function trackComplete() {
  vpPlugin.trackComplete();
  vpPlugin.trackVideoUnload();
}
```

The following example demonstrates event handling for a YouTube player:

```
function onYouTubePlayerReady(id) {
  player = document.getElementById("ytplayer");
  if (player.addEventListener) {
    player.addEventListener('onStateChange', 'handlePlayerStateChange');
  }
  else {
```

```
        player.attachEvent('onStateChange', 'handlePlayerStateChange');
    }
}

function handlePlayerStateChange (state) {
    switch (state) {
        case 1:
        case 3:
            // Video has begun playing/buffering
            if (player.getCurrentTime() == 0) {
                vpPlugin.trackVideoLoad();
                vpPlugin.trackPlay();
            } else {
                vpPlugin.trackPlay();
            }
            break;
        case 2:
            vpPlugin.trackPause();
        case 0:
            // Video has been paused/ended
            vpPlugin.trackComplete();
            vpPlugin.trackVideoUnload();
            break;
    }
}
```

Note that each player provides a different way to listen to events. Use the documentation provided by the player API to determine how to listen for player events.

Your next step is to [Test Your Video Measurement Code](#)

Track Methods and Player Events

Information about the correspondence between player events and the associated call exposed by the public API of the video heartbeat library.

The video player being instrumented must be capable of triggering a series of events through which any subscriber can be informed about what happens inside the video player. The following tables present the one-to-one correspondence between player events and the associated call exposed by the public API of the video heartbeat library.

This section contains the following information:

- [Video Playback](#)
- [Rules and Practices](#)
- [Ad Playback](#)
- [Chapter Tracking](#)
- [QoS Tracking](#)
- [Error Tracking](#)

Video Playback

Event	Method Call	Parameter List
Load the main video asset	<code>trackVideoLoad()</code>	None
Unload the main video asset	<code>trackVideoUnload()</code>	None
Autoplay ON, or user clicks play	<code>trackSessionStart()</code>	None
Playback start	<code>trackPlay()</code>	None
Playback stop/pause	<code>trackPause()</code>	None
Playback complete	<code>trackComplete()</code>	None
Seek start	<code>trackSeekStart()</code>	None
Seek complete	<code>trackSeekComplete()</code>	None
Buffer start	<code>trackBufferStart()</code>	None
Buffer complete	<code>trackBufferComplete()</code>	None

Rules and Practices

• Methods to be called in pairs:

The following methods must be called in pairs (that is, each `track...Start()` must have a corresponding `track...Complete()`):

- `trackBufferStart()` and `trackBufferComplete()`
- `trackPause()` and `trackPlay()` (note that if the player is closed before the pause resumes, the corresponding method might not be called)
- `trackSeekStart()` and `trackSeekComplete()` (with an exception: there may be multiple `trackSeekStart()` calls before a `trackSeekComplete()`)
- `trackAdStart()` and `trackAdComplete()` (unless the user seeks out of the ad without playing it to completion)
- `trackChapterStart()` and `trackChapterComplete()` (unless the user seeks out of the chapter without playing it to completion)

The `track...Start()` call is not required to be followed by a `track...Complete()` call, as there may be other `track...()` method calls in between. For example, the following sequence of `track...()` method calls is valid and describes a user who is seeking through the stream while paused, and resumes playback after two seeks:

```
trackPause(); // Signals that the user paused the playback.
trackSeekStart(); // Signals that the user started a seek operation.
trackSeekStart(); // Signals that the user started another seek operation (before the first
one was completed).
trackSeekComplete(); // Signals that the second seek operation has completed.
trackPlay(); // Signals that the user resumed playback.
```

• Tracking the completion of content:

The `trackComplete()` method is used to signal the completion of the video (i.e., the content was played to the end). You should call `trackComplete()` before calling `trackVideoUnload()` if the video was completed. When the user quits the video before its completion (e.g., by switching to another video in a playlist), you should not call `trackComplete()`. Instead, you should simply close the tracking session by calling `trackVideoUnload()`.

Ad Playback

Event	Method Call	Parameter List
An ad starts	<code>trackAdStart()</code>	None
An ad completes	<code>trackAdComplete()</code>	None

The `trackAdStart()` and `trackAdComplete()` methods are the only track methods required in order to signal the beginning and completion of an ad.

You do not need to (and should not) call any additional track methods to signal the transition from ad to content or vice-versa. For instance, you should not signal the pause of the main video (via `trackPause()`) when an ad starts. This is handled automatically by the `VideoPlayerPlugin` when you call `trackAdStart()`.

Chapter Tracking

Event	Method Call	Parameter List
A new chapter starts	<code>trackChapterStart()</code>	None
A chapter completes	<code>trackChapterComplete()</code>	None

QoS Tracking

Event	Method Call	Parameter List
A switch to another bitrate occurs	<code>trackBitrateChange()</code>	None

Error Tracking

Event	Method Call	Parameter List
An error occurs at the player level	<code>trackVideoPlayerError()</code>	String <code>errorId</code> - unique error identifier
An error occurs at the application level	<code>trackApplicationError()</code>	String <code>errorId</code>

Test Your Video Measurement Code

A simple way to test your video heartbeat implementation is to run the code in a demo environment.

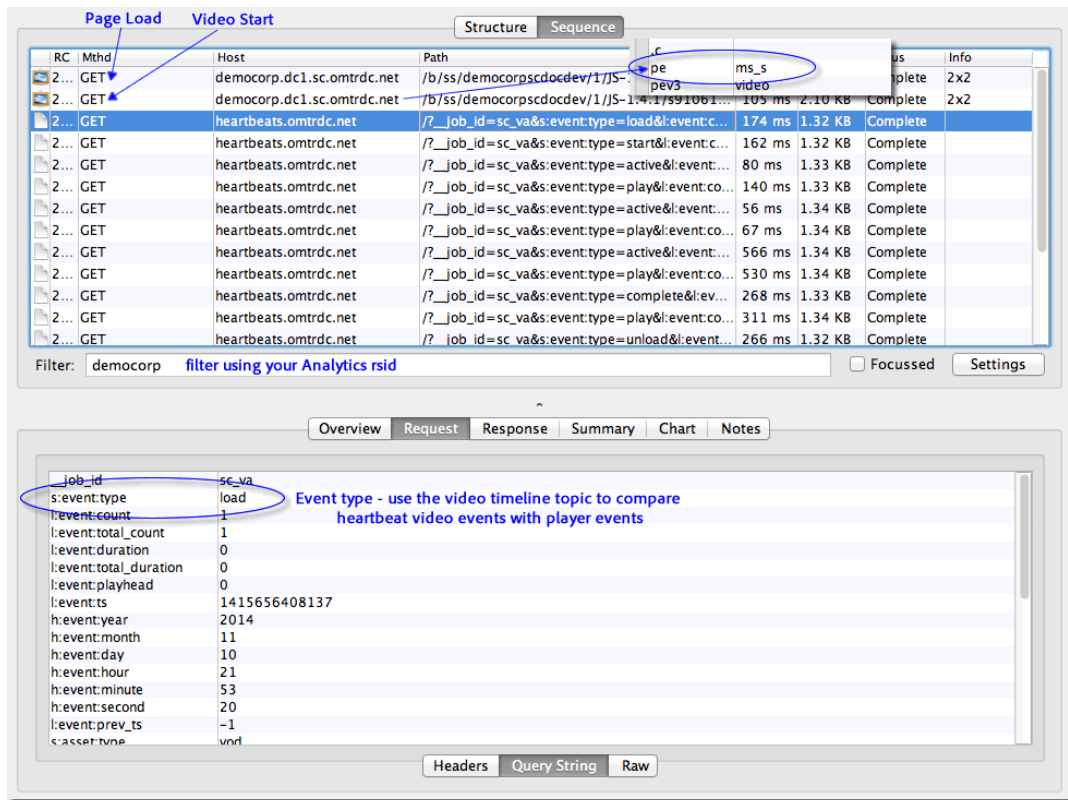
1. Load your code in a test environment and use a [packet analyzer](#) to verify that Analytics server calls and heartbeat calls are being sent. You should see an initial call to your data collection server, and then multiple calls to the Video Heartbeat tracking server.

In the initial call to your Analytics data collection server:

- Verify that `pe=ms_s`.

2. Test your implementation thoroughly to make sure you haven't missed any events. For example, if your player provides a pause event handler and you do not call `trackPause`, your time played metrics will be inflated.

- In a packet analyzer, inspect the calls and use the *Scenarios* to make sure events are being sent as expected. For example, you should see an `s:event:type` of `load` and then `start` when the video begins, and `complete` and then `unload` events when the video completes.



Video Measurement Parameters

List of data-collection parameters sent by video heartbeat.

This section contains the following information:

- [Video Core Parameters](#)
- [Video Ad Parameters](#)
- [Video Chapter Parameters](#)
- [Video Quality Parameters](#)
- [Other Parameters](#)

Video Core Parameters

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
Video Name	No	classification	<code>a.media.friendlyName</code>	N/A	Video Start
Video Length	Yes	classification	<code>a.media.length</code>	N/A	Video Start
Content	Yes	eVar	<code>a.media.name</code>	video	Video Start

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
Content Segment	Yes	eVar	a.media.segment	videosegment	Heartbeat
Content Type	Yes	eVar	a.contentType	videocontenttype	Video Start
Content Player Name	Yes	eVar	a.media.playerName	videoplayername	Video Start
Content Channel	No	eVar	a.media.channel	videochannel	Video Start
Video Initiates	Yes	event	a.media.view	videostart	Video Start
Content Starts	No	event	a.media.play	videoplay	Heartbeat
Content Completes	No	event	a.media.complete	videocomplete	Heartbeat
Content Segment Views	Yes	event	a.media.segmentView	videosegmentviews	Heartbeat
Content Time Spent	Yes	event	a.media.timePlayed	videotime	Heartbeat
Video Time Spent	Yes	event	a.media.totalTimePlayed	videototaltime	Heartbeat
10% Progress Marker	No	event	a.media.progress10	videoprogess10	Heartbeat
25% Progress Marker	No	event	a.media.progress25	videoprogess25	Heartbeat
50% Progress Marker	No	event	a.media.progress50	videoprogess50	Heartbeat
75% Progress Marker	No	event	a.media.progress75	videoprogess75	Heartbeat
95% Progress Marker	No	event	a.media.progress95	videoprogess95	Heartbeat
Average Minute Audience	No	event	a.media.averageMinuteAudience	videoaverageminuteaudience	Heartbeat
Video Path	Yes	prop	a.media.name	videopath	Video Start
Paused Impacted Streams	No	event	a.media.pause	videopause	Heartbeat
Pause Events	No	event	a.media.pauseCount	videopausecount	Heartbeat

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
Total Pause Duration	No	event	a.media.pauseTime	videopausetime	Heartbeat
Content Resumes	No	event	a.media.resume	videoresume	Heartbeat

Video Ad Parameters

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
Ad Name	No	classification	a.media.ad.friendlyName	N/A	Ad Start
Ad Length	Yes	classification	a.media.ad.length	N/A	Ad Start
Ad	Yes	eVar	a.media.ad.name	videoad	Ad Start
Pod Name	No	classification	a.media.ad.podFriendlyName	N/A	Ad Start
Pod Position	Yes	classification	a.media.ad.podSecond	N/A	Ad Start
Ad Pod	Yes	eVar	a.media.ad.pod	videoadpod	Ad Start
Ad in Pod Position	Yes	eVar	a.media.ad.podPosition	videoadinpod	Ad Start
Ad Player Name	Yes	eVar	a.media.ad.playerName	videoplayername	Ad Start
Ad Starts	Yes	event	a.media.ad.view	videostart	Ad Start
Ad Completes	Yes	event	a.media.ad.complete	videocomplete	Heartbeat
Ad Time Spent	Yes	event	a.media.ad.timePlayed	videoadtime	Heartbeat

Video Chapter Parameters

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
Chapter Name	No	classification	a.media.chapter.friendlyName	N/A	Heartbeat
Chapter Position	Yes	classification	a.media.chapter.position	N/A	Heartbeat
Chapter Offset	No	classification	a.media.chapter.offset	N/A	Heartbeat
Chapter Length	No	classification	a.media.chapter.length	N/A	Heartbeat
Chapter	Yes	eVar	a.media.chapter.name	videochapter	Heartbeat
Chapter Starts	Yes	event	a.media.chapter.view	videochapterstart	Heartbeat

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
Chapter Completes	No	event	a.media.chapter.complete	videochaptercomplete	Heartbeat
Chapter Time Spent	Yes	event	a.media.chapter.timePlayed	videochaptertime	Heartbeat

Video Quality Parameters

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
Time to Start	No	eVar	a.media.qoe.timeToStart	videoqoetimetostartevvar	Heartbeat
		event		videoqoetimetostart	
Buffer Events	No	eVar	a.media.qoe.bufferCount	videoqoebuffercountevvar	Heartbeat
		event		videoqoebuffercount	
Total Buffer Duration	No	eVar	a.media.qoe.bufferTime	videoqoebuffertimeevvar	Heartbeat
		event		videoqoebuffertime	
Bitrate Changes	No	eVar	a.media.qoe.bitrateChangeCount	videoqoebitratechangeevvar	Heartbeat
		event		videoqoebitratechange	
Average Bitrate	No	eVar	a.media.qoe.bitrateAverageBucket	videoqoebitrateaverageevvar	Heartbeat
Errors / Error Events	No	eVar	a.media.qoe.errorCount	videoqoeerrorcountevvar	Heartbeat
		event		videoqoeerrorcount	
Dropped Frames	No	eVar	a.media.qoe.droppedFrameCount	videoqoedroppedframeevvar	Heartbeat
		event		videoqoedroppedframecount	
Drops before Start	No	event	a.media.qoe.dropBeforeStart	videoqoedropbeforestart	Heartbeat
Buffer Impacted Streams	No	event	a.media.qoe.buffer	videoqoebuffer	Heartbeat
Bitrate Change Impacted Streams	No	event	a.media.qoe.bitrateChange	videoqoebitratechange	Heartbeat
Average Bitrate	No	event	a.media.qoe.bitrateAverage	videoqoebitrateaverage	Heartbeat

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
Error Impacted Streams	No	event	<code>a.media.qoe.error</code>	<code>videoqoeerror</code>	Heartbeat
Dropped Frame Impacted Streams	No	event	<code>a.media.qoe.droppedFrames</code>	<code>videoqoedroppedframes</code>	Heartbeat

Other Parameters

Label	Required?	Variable Type	Context Data Variable	Clickstream/API Variable Name	Sent With
SDK Version	No	N/A*	<code>a.media.sdkVersion</code>	N/A	Heartbeat
VHL Version	No	N/A*	<code>a.media.vhlVersion</code>	N/A	Heartbeat
Stalling Impacted Streams	No	N/A*	<code>a.media.qoe.stall</code>	N/A	Heartbeat
Stalling Events	No	N/A*	<code>a.media.qoe.stallCount</code>	N/A	Heartbeat
Total Stalling Duration	No	N/A*	<code>a.media.qoe.stallTime</code>	N/A	Heartbeat

* You must create your own processing rule if you want to use this parameter.

Sample player

Debugging

You can enable or disable logging for `MediaHeartbeat`.

Enable Debug Logging

You can enable or disable logging for each video heartbeat component.

The video heartbeat library provides an extensive tracing/logging mechanism that is put in place throughout the entire video-tracking stack. You can enable or disable this logging for each video heartbeat component by setting the `debugLogging` flag on the configuration object.

The log messages follow this format:

```
Format: [<timestamp>] [<level>] [<tag>] [<message>]
Example: [16:01:48 GMT+0200.848] [INFO]
```

```
[com.adobe.primetime.va.plugins.videoplayer::VideoPlayerPlugin] \  
  Data from delegate > ChapterInfo: name=First chapter, length=15, position=1, startTime=0
```

There are several sections delimited by pairs of square brackets as follows:

- **timestamp:** This is the current CPU time (time-zoned for GMT)
- **level:** There are 4 message levels defined:
 - INFO – Usually the input data from the application (validate player name, video ID, etc.)
 - DEBUG – Debug logs, used by the developers to debug more complex issues
 - WARN – Indicates potential integration/configuration errors or Heartbeats SDK bugs
 - ERROR – Indicates important integration errors or Heartbeats SDK bugs
- **tag:** The name of the sub-component that issued the log message (usually the class name)
- **message:** The actual trace message

You can use the logs output by the video heartbeat library to verify the implementation. A good strategy is to search through the logs for the string `#track`. This will highlight all the `track...()` APIs called by your application.

For instance, this is what the logs filtered for `#track` could look like:

```
[17:47:48 GMT+0200 (EET).942] [INFO] [plugin::player] #trackVideoLoad()  
[17:47:48 GMT+0200 (EET).945] [INFO] [plugin::player] #trackPlay()  
[17:47:48 GMT+0200 (EET).945] [INFO] [plugin::player] #trackPlay() > Tracking session auto-start.  
[17:47:48 GMT+0200 (EET).945] [INFO] [plugin::player] #trackSessionStart()  
[17:47:49 GMT+0200 (EET).446] [INFO] [plugin::player] #trackChapterStart()  
[17:47:49 GMT+0200 (EET).446] [INFO] [plugin::player] #trackChapterComplete()  
[17:48:10 GMT+0200 (EET).771] [INFO] [plugin::player] #trackComplete()  
[17:48:10 GMT+0200 (EET).774] [INFO] [plugin::player] #trackVideoUnload()
```

Using this validation method, you can easily spot implementation issues (e.g., the integration code never calls `trackAdComplete()` when an ad completes playback).

Validate implementations

To validate your Media Heartbeat implementation it will be required to use a HTTP Proxy tool to view the HTTP / HTTPS traffic between the Application and Heartbeats/Adobe Analytics.

HTTP calls for video analytics tracking will be sent to 2 different tracking servers:

- **Adobe Analytics:** Adobe Analytics hits are used to mark the initiate of a Video/Ad/Chapter. Tracking server example:
`<visitorsnamespace>.sc.omtrdc.net`

The different parameters related to video tracking for the Adobe Analytics HTTP calls are described in [Adobe Analytics parameters](#).

- **Heartbeats platform:** Heartbeat platform hits (also known as heartbeats) are sent throughout the video tracking session at 10 seconds intervals (out of band events might be sent outside of the 10 seconds cycle). Tracking server example:
`<visitorsnamespace>.hb.omtrdc.net`

The different parameters related to video tracking for the Adobe Analytics HTTP calls are described in [Heartbeats parameters](#).

Adobe Debug

Optionally, you can debug payloads (Heartbeat and Adobe Analytics) going out of Video Heartbeat Library using Adobe Debug tool which is a freely available tool from Adobe for Video Heartbeat customers.

To use Adobe Debug, you need to contact your Adobe representative for the initial setup and registration. After you gain access to Adobe Debug, go to [Adobe Debug help](#) to see the help information.

Heartbeats parameters

	Name	Required/Optional	Data Source	Description
All Events	s:event:type	R	Heartbeat SDK	The type of the event being tracked.
	l:event:prev_ts	R	Heartbeat SDK	The timestamp of the last event of the same type in this session. The value is -1 if this is the first event of this type in this video session.
	l:event:ts	R	Heartbeat SDK	The timestamp of the event.
	l:event:duration	R	Heartbeat SDK	
	l:event:playhead	R	VideoInfo object	The playhead is inside the currently active asset (main or ad), when the event was recorded
	s:event:sid	R	Heartbeat SDK	Randomly generated string, the session id. All events in a certain session (video + ads) should be the same
	l:asset:duration / l:asset:length	R	VideoInfo object	Video asset length of the main asset.
	s:asset:publisher	R	Adobe Heartbeat Plugin Config object	Publisher of the asset
	s:asset:video_id	R	VideoInfo object	ID uniquely identifying the video in the publisher's catalog
	s:asset:type	R	Heartbeat SDK	Asset type (main or ad).

	Name	Required/Optional	Data Source	Description
	s:stream:type	R	VideoInfo object	The stream type. Can be one of the following: live, vod, linear.
	s:user:id	O	Config object for mobile, app measurement VisitorID	User's specifically set visitor id
	s:user:aid	O		The user's analytics visitor id value.
	s:user:mid	R	Marketing Cloud Org	The user's marketing cloud visitor id value.
	s:cuser:customer_user_ids_x	O	AdobeAnalyticsPlugin	All customer user ids set on Audience Manager
	l:aam:loc_hint	R	AdobeAnalyticsPlugin	AAM data sent on each payload after aa_start
	s:aam:blob	R	AdobeAnalyticsPlugin	AAM data sent on each payload after aa_start
	s:sc:rsid	R	Report Suit ID (or ids)	SiteCatalyst RSID where reports should be sent
	s:sc:tracking_server	R	AdobeHeartbeatPluginConfig object	SiteCatalyst tracking server
	h:sc:ssl	R	AdobeHeartbeatPluginConfig object	Whether the traffic is over HTTPS (if set to 1) or over HTTP (is set to 0).
	s:sp:ovp	O	AdobeHeartbeatPluginConfig object	"primetime" for Primetime players, the actual OVP for other players

	Name	Required/Optional	Data Source	Description
	s:sp:sdk	R	Adobe Analytics Plugin Config object	OVP version string
	s:sp:player_name	R	VideoInfo object	Video player name (the actual player software, used to identify the player)
	s:sp:channel	O	Adobe Analytics Plugin Config object	The channel where the user is watching the content. For a mobile app, the app name. For a website, the domain name.
	s:sp:hb_version	R	Heartbeat SDK	The version number of the VideoHeartbeat library issuing the call.
	l:stream:bitrate	R	QosInfo object	The current value of the stream bitrate (in bps)
Error Events	s:event:source	R	Heartbeat SDK	The source of the error, either player-internal, or the application-level.
	s:event:id	R	s:event:id	Error id, uniquely identifies the error
s:asset:type=ad Events	s:asset:ad_id	R	AdInfo object	
	s:asset:ad_sid	R	Heartbeat SDK	
	s:asset:pod_id	R	Heartbeat SDK	Pod id inside the video. This value is computed automatically based on the following formula: MD5(video_id) + "_ " + index of the pod.

	Name	Required/Optional	Data Source	Description
	s:asset:pod_position	R	AdBreakInfo object	Index of the ad inside the pod (first ad has index 0, second ad index 1 etc.)
	s:asset:resolver	R	AdBreakInfo object	
	s:meta:custom_ad_metadata.x	O		Custom ad metadata
Chapter Events	s:stream:chapter_sid	R	Heartbeat SDK	Unique identifier associated to the playback instance of the chapter. Note: a chapter can be played multiple times due to seek-back operations performed by the user.
	s:stream:chapter_name	O	ChapterInfo object	The chapter's friendly (i.e. human readable) name.
	s:stream:chapter_id	R	Heartbeat SDK	The unique ID of the chapter. This value is computed automatically based on the following formula: MD5(video_id) + "_" + chapter_pos.
	l:stream:chapter_pos	R	ChapterInfo object	The chapter's index in the list of chapters (starting with 1).
	l:stream:chapter_offset	R	ChapterInfo object	The chapter's offset inside main content, excluding ads. (expressed in seconds)

	Name	Required/Optional	Data Source	Description
	l:stream:chapter_length	R	ChapterInfo object	The chapter's duration (expressed in seconds)
	s:meta:custom_chapter_metadata.x	O		Custom chapter metadata

Adobe Analytics parameters

Events	Name	Heartbeat Mapping Parameter	Required/optional	Value Range
Content	pe		R	ms_s
	pe3		R	video
	cid.customer_user_ids_x	scuser:customer_user_ids_x	O	
	c.a.contentType	s:asset:type	R	
	c.a.media.channel	s:sp:channel	R	
	c.a.media.playerName	s:sp:player_name	O	
	c.a.media.vsid	s:event:sid	R	
	c.a.media.view		R	TRUE
	c.a.media.name	s:asset:video_id		
	c.customer_video_metadata	s:meta:customer_video_metadata	O	
c.customer_chapter_metadata	s:meta:customer_chapter_metadata	O		
Ad	pe		R	msa_s
	pe3		R	videoAd
	cid.customer_user_ids_x	scuser:customer_user_ids_x	O	
	c.a.contentType	s:asset:type	R	

Events	Name	Heartbeat Mapping Parameter	Required/optional	Value Range
	c.a.media.channel	s:sp:channel		
	c.a.media.playerName	s:sp:player_name		
	c.a.media.vsid	s:event:sid	R	
	c.a.media.view		R	TRUE
	c.a.media.name	s:asset:video_id	R	
	c.a.ad.media.name	s:asset:ad_id	R	
	c.a.ad.media.playerName			
	c.a.ad.media.pod	s:asset:pod_id		
	c.a.ad.media.podPosition	s:asset:position		
	c.a.ad.media.view			TRUE
	c.customer_video_metadata	s:metadata:customer_video_metadata	O	
	c.customer_ad_metadata	s:metadata:customer_ad_metadata	O	

Ratings Partners Integration

(Note: Certified Metrics description including certification, contract, etc included in “home page”)

Partner	Documentation
Nielsen	Digital Content Ratings powered by Adobe
comScore	Certified Metrics powered by Adobe

Scenarios

This topic provides a scenario to illustrate when video data is collected.

Scenario and Timeline Illustrations

This topic describes a scenario to illustrate when video data is collected and contains illustrations to show the video and actions timelines.

This section contains the following information:

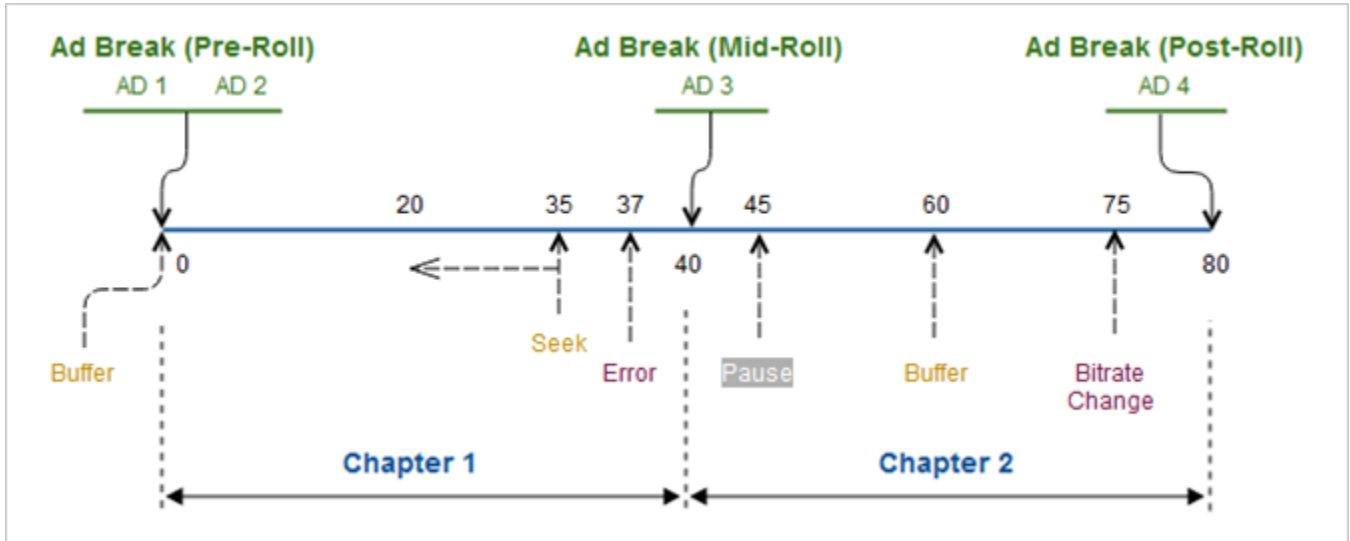
- [Scenario Overview](#)
- [Video Timeline](#)
- [Actions Timeline](#)

Scenario Overview

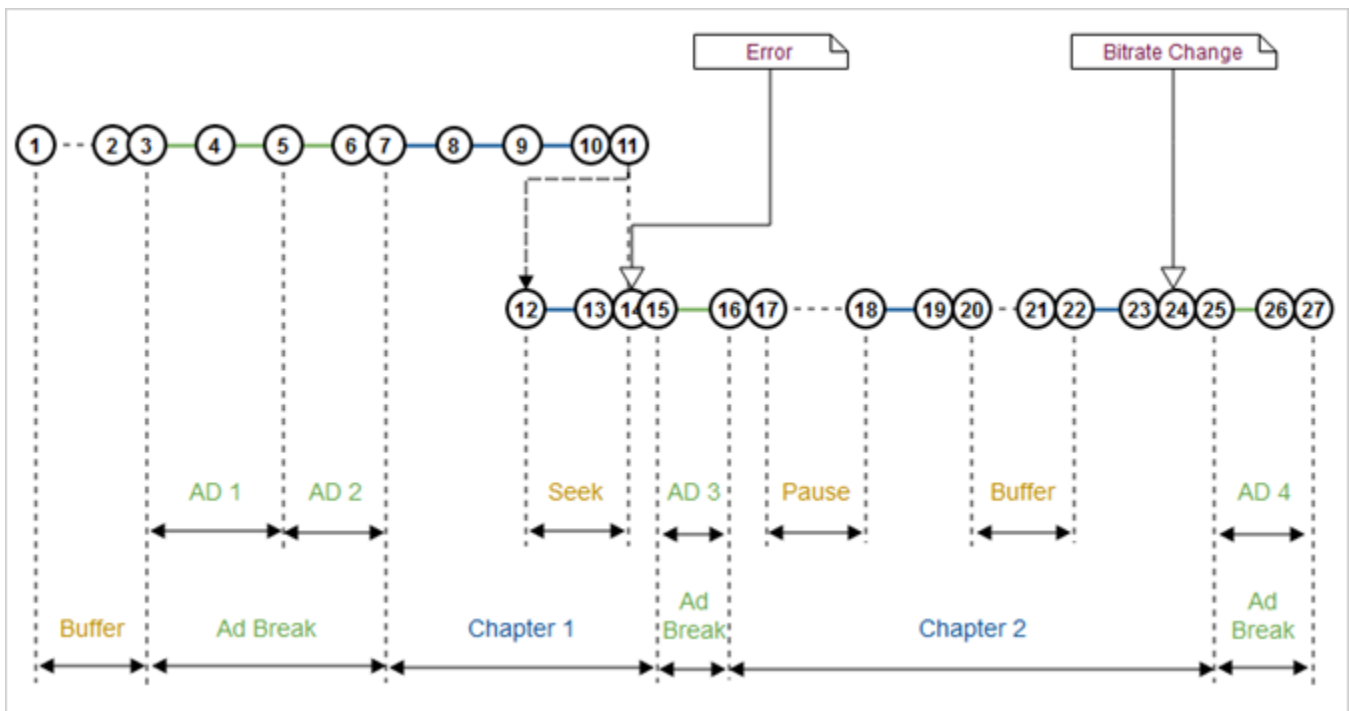
A video (VOD) is loaded and played into a web page or application that has the following components:

Component	Details
Playback content	<p>Main content of 80 seconds split in two chapters.</p> <ul style="list-style-type: none">• Chapter 1 - chapter duration: 40 seconds• Chapter 2 - chapter duration: 40 seconds <p>Three ad breaks:</p> <ul style="list-style-type: none">• One pre-roll before first chapter that contains two ads:<ul style="list-style-type: none">• AD 1 - ad duration: 20 seconds• AD 2 - ad duration: 15 seconds• One mid-roll between chapters that contains one ad:<ul style="list-style-type: none">• AD 3 - ad duration: 10 seconds• One post-roll at the end of the content that contains one ad:<ul style="list-style-type: none">• AD 4 - ad duration: 15 seconds
User interactions	<ul style="list-style-type: none">• Start the content after it is loaded.• Skip back 15 seconds of content inside Chapter 1 at second 35.• Pause the main content for 45 seconds during Chapter 2.
Playback events	<ul style="list-style-type: none">• Buffering on start for 15 seconds.• Player error occurs during Chapter 1 at second 37.• Re-buffering for 15 seconds during Chapter 2 at second 60.• Bitrate changed during Chapter 2 at second 75.

Video Timeline



Actions Timeline




Tracking Explained

This topic describes when video data is collected and contains information about the actions a user takes along with video heartbeat Library methods used, Analytics and video heartbeat library calls made, and implementation details.

See [Scenario and Timeline Illustrations](#) to view illustrations depicting the processes explained below.

The scenario illustrated in the following table is a typical end-to-end playback where there is little interaction and content is played to the end.

 **Note:** *VideoPlayerPluginDelegate* must provide the most up-to-date information it has when queried: *VideoInfo* (including *playhead*), *AdBreakInfo*, *AdInfo*, *ChapterInfo*, and *QoSInfo*.

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
1	0	0	<p>Actions: Auto-play or Play button pressed</p> <hr/> <p>Video Heartbeat Library: trackVideoLoad trackSessionStart</p> <hr/> <p>Analytics Tracking Calls: SC Video Start Call</p> <hr/> <p>Video Heartbeat Tracking Calls: HB start event HB AA start event</p> <hr/> <p>Implementation Details:</p> <ul style="list-style-type: none"> • Start the tracking library internal session by calling trackVideoLoad • Set VideoInfo before any tracking method is called • Start tracking the startup time by calling trackSessionStart method

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
2	10	0	<p>Actions: N/A</p> <hr/> <p>Video Heartbeat Library: N/A</p> <hr/> <p>Analytics Tracking Calls: N/A</p> <hr/> <p>Video Heartbeat Tracking Calls: HB start event</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			<p>Implementation Details:</p> <p>This call is sent because the app takes longer than 10 seconds to start the stream (long buffering scenario).</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
3	15	0	<p>Actions:</p> <p>Ad start (AD1)</p> <hr/> <p>Video Heartbeat Library:</p> <p>trackAdStart</p> <p>trackPlay</p> <hr/> <p>Analytics Tracking Calls:</p> <p>SC Ad Start Call</p> <hr/> <p>Video Heartbeat Tracking Calls:</p> <p>HB start event</p> <p>HB ad start event</p> <p>HB AA ad start event</p> <p>HB play event</p> <hr/> <p>Implementation Details:</p> <ul style="list-style-type: none"> • Set <code>AdBreakInfo</code> before the <code>trackAdStart</code> method is called for the first ad on the current ad break (pre-roll). • Set <code>AdBreakInfo.position</code> to 1 because the first ad break is inside the current main content. • Set <code>AdBreakInfo.startTime</code> to 0. The <code>startTime</code> is the offset in the main content (in seconds) where the ad break starts. This can also be seen as the value of the playhead when the ad break is reached. • Set <code>AdInfo</code> for AD 1 before the <code>trackAdStart</code> method is called. • Set <code>AdInfo.position</code> to 1 for AD 1 because the first ad is inside the current ad break.

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
4	25	0	<p>Actions: N/A</p> <hr/> <p>Video Heartbeat Library: N/A</p> <hr/> <p>Analytics Tracking Calls: N/A</p> <hr/> <p>Video Heartbeat Tracking Calls: HB ad play event</p> <hr/> <p>Implementation Details: N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
5	35	0	<p>Actions: Ad complete (AD1) Ad start (AD2)</p> <hr/> <p>Video Heartbeat Library: trackAdComplete trackAdStart</p> <hr/> <p>Analytics Tracking Calls: SC Ad Start Call</p> <hr/> <p>Video Heartbeat Tracking Calls: HB ad play event HB ad complete event HB ad start event HB AA ad start event</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			<p>Implementation Details:</p> <ul style="list-style-type: none"> • Set <code>AdInfo</code> to NULL for AD 1 after the <code>trackAdComplete</code> method is called. • Set <code>AdInfo</code> for AD 2 before the <code>trackAdStart</code> method is called. • Set <code>AdInfo.position</code> to 2 for AD 2 because the second ad is inside current ad break.

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
6	45	0	<p>Actions: N/A</p> <hr/> <p>Video Heartbeat Library: N/A</p> <hr/> <p>Analytics Tracking Calls: N/A</p> <hr/> <p>Video Heartbeat Tracking Calls: HB ad play event</p> <hr/> <p>Implementation Details: N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
7	50	0	<p>Actions: Ad complete (AD2)</p> <hr/> <p>Video Heartbeat Library: <code>trackAdComplete</code> <code>trackChapterStart</code></p> <hr/> <p>Analytics Tracking Calls:</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			N/A
			<p>Video Heartbeat Tracking Calls:</p> <p>HB ad play event</p> <p>HB ad complete event</p> <p>HB chapter start event</p> <p>HB play event</p>
			<p>Implementation Details:</p> <ul style="list-style-type: none"> • Set <code>AdInfo</code> to NULL for AD 2 after the <code>trackAdComplete</code> method is called. • Set <code>AdBreakInfo</code> to Null for the current ad break (pre-roll) after the <code>trackAdComplete</code> method is called. • Set <code>ChapterInfo</code> for Chapter 1 before the <code>trackChapterStart</code> method is called.

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
8	60	10	<p>Actions:</p> <p>N/A</p>
			<p>Video Heartbeat Library:</p> <p>N/A</p>
			<p>Analytics Tracking Calls:</p> <p>N/A</p>
			<p>Video Heartbeat Tracking Calls:</p> <p>HB play event</p>
			<p>Implementation Details:</p> <p>N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
9	70	20	Actions: N/A
			Video Heartbeat Library: N/A
			Analytics Tracking Calls: N/A
			Video Heartbeat Tracking Calls: HB play event
			Implementation Details: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
10	80	30	Actions: N/A
			Video Heartbeat Library: N/A
			Analytics Tracking Calls: N/A
			Video Heartbeat Tracking Calls: HB play event
			Implementation Details: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
11	85	35	<p>Actions: Seek Back 15"</p> <hr/> <p>Video Heartbeat Library: trackSeekStart</p> <hr/> <p>Analytics Tracking Calls: N/A</p> <hr/> <p>Video Heartbeat Tracking Calls: HB play event</p> <hr/> <p>Implementation Details: N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
12	85	20	<p>Actions: N/A</p> <hr/> <p>Video Heartbeat Library: trackSeekComplete</p> <hr/> <p>Analytics Tracking Calls: N/A</p> <hr/> <p>Video Heartbeat Tracking Calls: HB play event</p> <hr/> <p>Implementation Details: Make sure <code>VideoPlayerPluginDelegate</code> will report the new playhead (20) after <code>trackSeekComplete</code> is called.</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
13	95	30	Actions: N/A
			Video Heartbeat Library: N/A
			Analytics Tracking Calls: N/A
			Video Heartbeat Tracking Calls: HB play event
			Implementation Details: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
14	102	37	Actions: Player error occurred
			Video Heartbeat Library: <code>trackError</code>
			Analytics Tracking Calls: N/A
			Video Heartbeat Tracking Calls: HB error event
			Implementation Details: <ul style="list-style-type: none"> • Set error type and message on <code>trackError</code> method.

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
15	105	40	<p>Actions:</p> <p>Ad start (AD3)</p> <hr/> <p>Video Heartbeat Library:</p> <p>trackChapterComplete</p> <p>trackAdStart</p> <hr/> <p>Analytics Tracking Calls:</p> <p>SC Ad Start Call</p> <hr/> <p>Video Heartbeat Tracking Calls:</p> <p>HB play event</p> <p>HB chapter complete event</p> <p>HB ad start event</p> <p>HB AA ad start event</p> <hr/> <p>Implementation Details:</p> <ul style="list-style-type: none"> • Set <code>ChapterInfo</code> to NULL for Chapter 1 after the <code>trackChapterComplete</code> method is called. • Set <code>AdBreakInfo</code> before the <code>trackAdStart</code> method is called for the first ad on the current ad break (mid-roll). <ul style="list-style-type: none"> • Set <code>AdBreakInfo.position</code> to 2 because the second ad break is inside the current main content. • Set <code>AdBreakInfo.startTime</code> to 40. The <code>startTime</code> is the offset in the main content (in seconds) where the ad break starts. This can also be seen as the value of the playhead when the ad break is reached. • set <code>AdInfo</code> for AD 3 before the <code>trackAdStart</code> method is called. <ul style="list-style-type: none"> • Set <code>AdInfo.position</code> to 1 for AD 3 because the first ad is inside the current ad break.

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
16	115	40	<p>Actions:</p> <p>Ad complete (AD3)</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			<p>Video Heartbeat Library:</p> <p>trackAdComplete</p> <p>trackChapterStart</p> <hr/> <p>Analytics Tracking Calls:</p> <p>N/A</p> <hr/> <p>Video Heartbeat Tracking Calls:</p> <p>HB ad play event</p> <p>HB ad complete event</p> <p>HB chapter start event</p> <p>HB play event</p> <hr/> <p>Implementation Details:</p> <ul style="list-style-type: none"> • Set <code>AdInfo</code> to NULL for AD 3 after the <code>trackAdComplete</code> method is called. • Set <code>AdBreakInfo</code> to Null for the current ad break (mid-roll) after the <code>trackAdComplete</code> method is called. • Set <code>ChapterInfo</code> for Chapter 2 before the <code>trackChapterStart</code> method is called.

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
17	120	45	<p>Actions:</p> <p>Pause button is pressed</p> <hr/> <p>Video Heartbeat Library:</p> <p>trackPause</p> <hr/> <p>Analytics Tracking Calls:</p> <p>N/A</p> <hr/> <p>Video Heartbeat Tracking Calls:</p> <p>HB play event</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			HB pause event
			Implementation Details: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
18	135	45	Actions: Play button is pressed after 15"
			Video Heartbeat Library: trackPlay
			Analytics Tracking Calls: N/A
			Video Heartbeat Tracking Calls: HB play event
			Implementation Details: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
19	145	55	Actions: N/A
			Video Heartbeat Library: N/A
			Analytics Tracking Calls: N/A
			Video Heartbeat Tracking Calls:

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			HB play event
			Implementation Details: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
20	150	60	Actions: Buffer start event occurred
			Video Heartbeat Library: trackBufferStart
			Analytics Tracking Calls: N/A
			Video Heartbeat Tracking Calls: HB play event HB buffer event
			Implementation Details: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
21	160	60	Actions: N/A
			Video Heartbeat Library: N/A
			Analytics Tracking Calls: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			<p>Video Heartbeat Tracking Calls: HB buffer event</p>
			<p>Implementation Details: N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
22	165	60	<p>Actions: Buffer end event occurred after 15"</p>
			<p>Video Heartbeat Library: trackBufferComplete</p>
			<p>Analytics Tracking Calls: N/A</p>
			<p>Video Heartbeat Tracking Calls: HB buffer event HB play event</p>
			<p>Implementation Details: N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
23	175	70	<p>Actions: N/A</p>
			<p>Video Heartbeat Library: N/A</p>
			<p>Analytics Tracking Calls: N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			<p>Video Heartbeat Tracking Calls:</p> <p>HB play event</p>
			<p>Implementation Details:</p> <p>N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
24	180	75	<p>Actions:</p> <p>Bitrate change occurred</p>
			<p>Video Heartbeat Library:</p> <p>trackBitrateChange</p>
			<p>Analytics Tracking Calls:</p> <p>N/A</p>
			<p>Video Heartbeat Tracking Calls:</p> <p>HB bitrate change event</p>
			<p>Implementation Details:</p> <p>N/A</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
25	185	80	<p>Actions:</p> <p>Ad start (AD4)</p>
			<p>Video Heartbeat Library:</p> <p>trackChapterComplete</p> <p>trackAdStart</p>
			<p>Analytics Tracking Calls:</p> <p>SC Ad Start Call</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			<p>Video Heartbeat Tracking Calls:</p> <p>HB play event</p> <p>HB chapter complete event</p> <p>HB ad start event</p> <p>HB AA ad start event</p> <hr/> <p>Implementation Details:</p> <ul style="list-style-type: none"> • Set <code>ChapterInfo</code> to NULL for Chapter 2 after the <code>trackChapterComplete</code> method is called. • Set <code>AdBreakInfo</code> before the <code>trackAdStart</code> method is called for the first ad on the current ad break (post-roll). <ul style="list-style-type: none"> • Set <code>AdBreakInfo.position</code> to 3 because the third ad break is inside the current main content. • Set <code>AdBreakInfo.startTime</code> to 80. The <code>startTime</code> is the offset in the main content (in seconds) where the ad break starts. This can also be seen as the value of the playhead when the ad break is reached. • Set <code>AdInfo</code> for AD 4 before the <code>trackAdStart</code> method is called. • Set <code>AdInfo.position</code> to 1 for AD 4 because the first ad is inside current ad break.

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
26	195	80	<p>Actions:</p> <p>N/A</p> <hr/> <p>Video Heartbeat Library:</p> <p>N/A</p> <hr/> <p>Analytics Tracking Calls:</p> <p>N/A</p> <hr/> <p>Video Heartbeat Tracking Calls:</p> <p>HB play event</p>

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
			Implementation Details: N/A

Action #	Actions Timeline (seconds)	Main Content Timeline (seconds)	
27	200	80	Actions: Ad complete (AD4)
			Video Heartbeat Library: trackAdComplete trackComplete trackUnload
			Analytics Tracking Calls: N/A
			Video Heartbeat Tracking Calls: HB play event HB ad complete event HB complete event
			Implementation Details: <ul style="list-style-type: none"> • Set <code>AdInfo</code> to NULL for AD 4 after the <code>trackAdComplete</code> method is called. • Set <code>AdBreakInfo</code> to Null for the current ad break (post-roll) after the <code>trackAdComplete</code> method is called. • Close the main content by calling <code>trackComplete</code>. • Close the tracking library internal session by calling <code>trackUnload</code>. • Destroy the heartbeat library instance calling the <code>destroy</code> method.

Non-Linear Tracking Scenarios

This topic describes when video data is collected in non-linear scenarios and contains information about the actions a user takes along with video heartbeat Library methods used, Analytics and video heartbeat library calls made, and implementation details.

The scenario illustrated in *Tracking Explained* is a typical end-to-end playback where there is little interaction and content is played to the end.

The following table illustrates tracking scenarios where the user seeks around more or drops from the stream.

Use Case	Scenario	Video Heartbeat Library	Implementation Details
Skip to next video	<ul style="list-style-type: none"> • playlist of videos, no ads, no chapters 	<ul style="list-style-type: none"> • trackVideoLoad • trackSessionStart (optional) • trackPlay <p>User clicks Next.</p> <ul style="list-style-type: none"> • trackVideoUnload • ... • trackVideoLoad • ... 	You can reuse one VHL instance but make sure to call trackVideoUnload/trackVideoLoad and update the player delegate videoInfo between the two clips.
Chapter seek	<ul style="list-style-type: none"> • one video content • 3 chapters • seek from chapter 1 to chapter 3 	<ul style="list-style-type: none"> • trackVideoLoad • trackSessionStart (optional) • tracktrackChapterStart <p>User seeks forward.</p> <ul style="list-style-type: none"> • trackSeekStart • trackSeekComplete • tracktrackChapterStart • tracktrackChapterComplete • trackComplete • trackVideoUnload • destroy 	When the user starts seeking, wait until it completes then call trackChapterStart with the new chapter info. You should NOT call trackChapterComplete on the source chapter, because it was not seen through the end.

Pause tracking

The Pause tracking support was added on VHL 1.6. At the same time, the buffer and pause behaviors were unified to have the same way of tracking and same metrics. VHL will send a new pause Video Heartbeat event at each 10 seconds during pause and will stop after 30 minutes, at this point the session will be closed. If the playback is resumed after 30 minutes of pause, VHL will automatically create a new tracking session and send a resume event.

For the implementation that are not passing to VHL all information about the player state, a new artificial tracking event was build called "stall" to define a state of the player that VHL does not know about. One simple example is when the player is in buffer mode, the playhead has same value during that period but the VHL was not informed due to the fact that the player is not exposing the event or just because the buffer event was not properly instrumented. The stalling event is tracked in the same way as paused.

Pause duration less than 30 minutes

In this scenario, complete the following tasks:

1. Start playback for a content that is 10 minutes long.

2. Pause the playback after 3 minutes.
3. Resume the content after 20 minutes and play the content until the end.



Expected events

- An Analytics `video initiate` event after the session starts.
- A `video heartbeats start` event after the session starts.
- `video heartbeats play` events every 10 seconds until the session is paused.
- `video heartbeats pause` events every 10 seconds while the session is paused.
- `video heartbeats play` events every 10 seconds after pause;
- A `video heartbeats complete` event when the playback is complete.
- A `complete video` call sent to Analytics when a session has ended.

Add pause metrics to analytics: has paused and number of pauses.

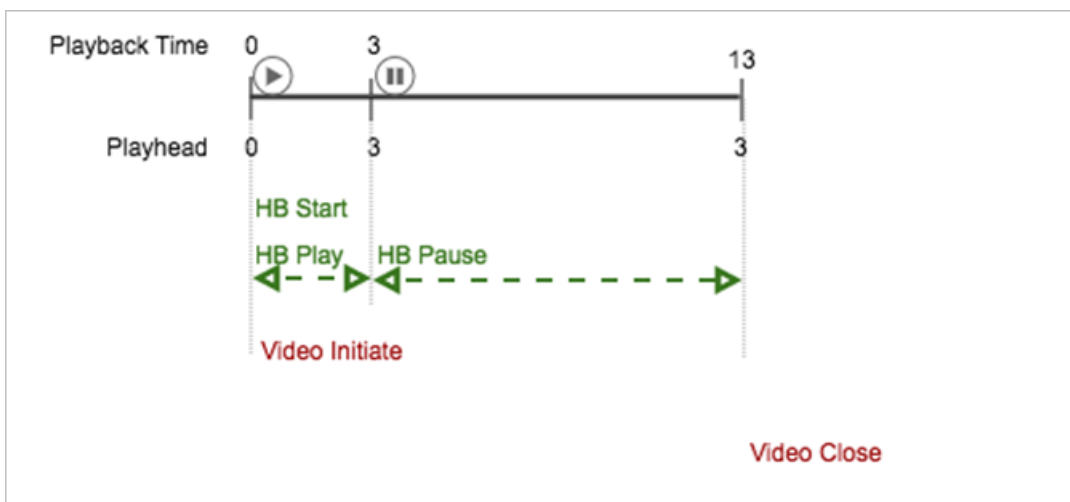
Expected metrics

- 1 content start and 10 minutes of total time spent.
- All video solution events + has pause event, 1 pause event count.

Abandon during pause

In this scenario, complete the following tasks:

1. Start playback for a content that is 10 minutes long.
2. Pause the playback after 3 minutes;
3. Close the content after 10 minutes.



Expected results

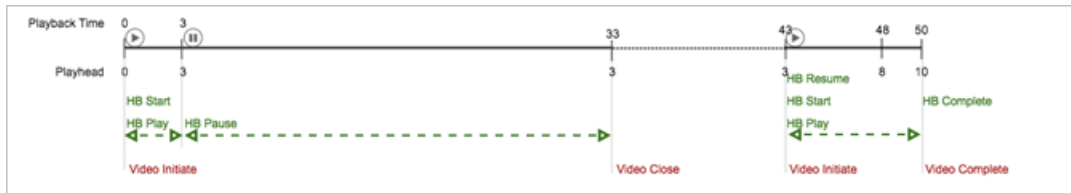
- An Analytics `video initiate` event when the session starts.
- A video heartbeats `start` event when the session starts.
- video heartbeats `play` events every 10 seconds until the session is paused.
- video heartbeats `pause` events every 10 seconds during the pause.
- A `close video` call sent to Analytics when a session has ended.

Add pause metrics to analytics: has paused and number of pauses.

Pause duration more than 30 minutes

In this scenario, complete the following tasks:

1. start playback for a content that is 10 minutes long.
2. pause the playback after 3 minutes.
3. resume the content after 40 minutes and play the content to the end.



Expected events

- An Analytics `video initiate` event after the session starts.
- A video heartbeats `start` event when the session starts.
- Video heartbeats `play` events every 10 seconds until the content is paused.
- Video heartbeats `pause` events every 10 seconds during pause for 30 minutes.
- A `close video` call sent to Analytics once a session has ended.

Add pause metrics to analytics: has paused and number of pauses.

- No video heartbeats event for 10 minutes until playback resumes.
- An Analytics `video initiate` event after a new tracking session starts after the playback resumes (including a new SID).
- A video heartbeats `start` event after the session starts (playback resumes).
- A video heartbeats `resume` event after the session starts (playback resumes).
- Video heartbeats `play` events every 10 seconds.
- A video heartbeats `complete` event after the playback is complete.
- A `complete video` call sent to Analytics after a session has ended.

Add resume metric to analytics: has resume.

Expected metrics

- 1 content starts and 3 minutes of total time spent.
- All video solution events + has pause event, and 1 pause event count.
- 1 content starts and 7 minutes of total time spent.
- All video solution events + has resume event.

Contact and Legal Information

Information to help you contact Adobe and to understand the legal issues concerning your use of this product and documentation.

Help & Technical Support

The Adobe Experience Cloud Customer Care team is here to assist you and provides a number of mechanisms by which they can be engaged:

- [Check the Marketing Cloud help pages for advice, tips, and FAQs](#)
- [Ask us a quick question on Twitter @AdobeExpCare](#)
- [Log an incident in our customer portal](#)
- [Contact the Customer Care team directly](#)
- [Check availability and status of Marketing Cloud Solutions](#)

Service, Capability & Billing

Dependent on your solution configuration, some options described in this documentation might not be available to you. As each account is unique, please refer to your contract for pricing, due dates, terms, and conditions. If you would like to add to or otherwise change your service level, or if you have questions regarding your current service, please contact your Account Manager.

Feedback

We welcome any suggestions or feedback regarding this solution. Enhancement ideas and suggestions [can be added to our Customer Idea Exchange](#).

Legal

© 2017 Adobe Systems Incorporated. All Rights Reserved.
Published by Adobe Systems Incorporated.

[Terms of Use](#) | [Privacy Center](#)

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. A trademark symbol (®, ™, etc.) denotes an Adobe trademark.

All third-party trademarks are the property of their respective owners. Updated Information/Additional Third Party Code Information available at <http://www.adobe.com/go/thirdparty>.