<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track buffering</td>
<td>82</td>
</tr>
<tr>
<td>Track seeking</td>
<td>85</td>
</tr>
<tr>
<td>Implement standard metadata</td>
<td>87</td>
</tr>
<tr>
<td>Track ads</td>
<td>96</td>
</tr>
<tr>
<td>Overview</td>
<td>96</td>
</tr>
<tr>
<td>Track ads on Android</td>
<td>99</td>
</tr>
<tr>
<td>Track ads on iOS</td>
<td>101</td>
</tr>
<tr>
<td>Track ads on JavaScript</td>
<td>103</td>
</tr>
<tr>
<td>Track ads on Chromecast</td>
<td>105</td>
</tr>
<tr>
<td>Track ads on Roku</td>
<td>106</td>
</tr>
<tr>
<td>Implement standard ad metadata</td>
<td>108</td>
</tr>
<tr>
<td>Track chapters and segments</td>
<td>109</td>
</tr>
<tr>
<td>Overview</td>
<td>109</td>
</tr>
<tr>
<td>Track chapters and segments on Android</td>
<td>111</td>
</tr>
<tr>
<td>Track chapters and segments on iOS</td>
<td>112</td>
</tr>
<tr>
<td>Track chapters and segments on JavaScript</td>
<td>113</td>
</tr>
<tr>
<td>Track chapters and segments on Chromecast</td>
<td>114</td>
</tr>
<tr>
<td>Track chapters and segments on Roku</td>
<td>115</td>
</tr>
<tr>
<td>Track quality of experience</td>
<td>116</td>
</tr>
<tr>
<td>Overview</td>
<td>116</td>
</tr>
<tr>
<td>Track quality of experience on Android</td>
<td>117</td>
</tr>
<tr>
<td>Track quality of experience on iOS</td>
<td>118</td>
</tr>
<tr>
<td>Track quality of experience on JavaScript</td>
<td>119</td>
</tr>
<tr>
<td>Track quality of experience on Chromecast</td>
<td>120</td>
</tr>
<tr>
<td>Track quality of experience on Roku</td>
<td>121</td>
</tr>
<tr>
<td>Track errors</td>
<td>122</td>
</tr>
<tr>
<td>Overview</td>
<td>122</td>
</tr>
<tr>
<td>Track errors on Android</td>
<td>123</td>
</tr>
<tr>
<td>Track errors on iOS</td>
<td>123</td>
</tr>
<tr>
<td>Track errors on JavaScript</td>
<td>123</td>
</tr>
<tr>
<td>Track errors on Chromecast</td>
<td>124</td>
</tr>
<tr>
<td>Track errors on Roku</td>
<td>124</td>
</tr>
<tr>
<td>Opt-out and privacy</td>
<td>124</td>
</tr>
<tr>
<td>Tracking scenarios</td>
<td>126</td>
</tr>
</tbody>
</table>
Media Collection API ................................................................................................................. 245
Overview ...................................................................................................................................... 245
API reference ............................................................................................................................... 248
  Sessions request ....................................................................................................................... 248
  Events request ....................................................................................................................... 249
  Request parameters ............................................................................................................... 250
  Event types and descriptions ................................................................................................. 255
  JSON validation schemas ....................................................................................................... 256
Implementing the API ................................................................................................................. 256
  Quick start ............................................................................................................................. 256
  Setting the HTTP request type in your player ....................................................................... 257
  Obtaining a session ID .......................................................................................................... 257
  Implementing an events request ............................................................................................ 258
  Validating event requests ...................................................................................................... 258
  Sending ping events ............................................................................................................. 259
  Sending QoE data .................................................................................................................. 259
  Custom metadata support ...................................................................................................... 259
  Timeout conditions ............................................................................................................... 260
  Controlling the order of events ............................................................................................ 260
  Queueing events when sessions response is slow ............................................................... 260
Media tracking timelines .......................................................................................................... 262
  Timeline 1 - View to end of content ..................................................................................... 262
  Timeline 2 - User abandons session .................................................................................... 269
  Timeline 3 - Chapters .......................................................................................................... 273
Track downloaded content ....................................................................................................... 280

Metrics and metadata ................................................................................................................. 283
  Audio and video parameters ................................................................................................. 283
  Ad parameters ....................................................................................................................... 304
  Chapter parameters ............................................................................................................. 311
  Quality parameters .............................................................................................................. 315
  Segments ............................................................................................................................... 326
  Calculated metrics ............................................................................................................... 327
Reporting and analysis ................................................................. 328
  Media reports enablement ......................................................... 328
  Media default reports ............................................................... 330
    Default reports overview ...................................................... 330
    Media overview .................................................................. 331
    Media detail ....................................................................... 332
    Media daypart .................................................................... 333
    Media concurrent viewers .................................................... 333
    Get concurrent viewers JSON report data ......................... 334
  Media workspace templates .................................................... 335

Federated Analytics ................................................................. 337

Additional Resources .............................................................. 339
  Documentation updates ......................................................... 339
  Contact and Legal Information .............................................. 340
Measuring audio and video in Adobe Analytics

This topic contains the following information:

- **Overview**
- **Benefits**
- **Heartbeat versus Milestone Benefits**
- **Devices Supported**

⚠️ **Important:** The documentation provided here is specific to clients utilizing version 1.5 or higher of Adobe’s Media SDK for heartbeat measurement, or Adobe’s newer Media Collection API for heartbeat measurement. It does not include instructions around the legacy milestone video implementation. We encourage all customers to move towards adopting one or both of the two latest media tracking solutions, in order to capitalize on improvements and expanded measurement. You can view the benefits of transitioning to the latest solutions below. While we will continue to support the milestone method of tracking videos, there will not be any planned updates, fixes, or feature improvements. Please reach out to your Adobe Account Manager if you have any further questions.

**Overview**

Adobe Analytics for Media (also referred to as Media Analytics) is an add-on to the base Analytics offering that provides clients with robust media measurement for content, audio and advertisements. Media Analytics provides many benefits to customers to allow for real-time monitoring, detailed analysis, actionable insights and monetization opportunities.

Media tracking is enabled through either of the following:

- **Media SDK** - Integrates with the most commonly used media players.
- **Media Collection API** - (RESTful API) Integrates with players for which there is no SDK support (or with players for which no SDK integration is desired).

The Media Collection API also provides an additional capability that is not yet available in the SDK:

- **Downloaded content tracking** - Provides support for tracking media content (video and audio) that is downloaded and played from a device regardless of the connectivity. This capability is built on top of the Media Collection API and follows the same player tracking specification. (There is no SDK support at this time.)

Adobe Analytics for Media enables clients to track the full customer journey across their site, which includes media consumption, and these measures are easily integrated into Analytics reporting and other Experience Cloud products. Media measurement allows you to slice and dice your data into multiple dimensions and segments, capturing all of the metadata you need to do a full detailed analysis, and to attribute success criteria to fully consumed media, average time spent, and completed ads.

The media solutions not only measure vital delivery metrics related to QoS, such as dropped frames, time spent buffering, and average bitrate. They can also be combined with your website or app data to visualize the flow of the customer and their interests, to better be able to make recommendations and personalize their experiences through the Adobe Experience Cloud.

**Benefits**

Some of the many benefits that Adobe’s media measurement solutions provide include:
• **Timely analysis** - Make real-time, actionable decisions utilizing key performance metrics (e.g., duration) across multiple channels. Main content events are measured in **10-second** intervals to capture all activity as it occurs. Ad tracking events occur at **1-second** intervals.

• **Drive engagement** - Fully engage users through fewer buffering events and by understanding where and when ads should play within content to provide a smooth, less intrusive experience that brings users back and delivers repeat visits.

• **Holistic picture** - Combine multiple data points across all of your content distributors to get a full view of all your media activity, and measure engagement and views/listens across all possible channels through **Federated Analytics**.

• **Increased granularity** - Evaluate viewing behavior at the most granular level, including individual visitor time of day, concurrent viewers/listeners by minute, and average duration the content was consumed.

• **Precise measurement** - Measure across the multiple devices used for media consumption, including OTT, smartphone, tablet, desktop, and more, to monitor user engagement patterns and habits.

• **Segmentation** - Apply classifications to your players, devices, genres, chapters, and shows to see how each has an impact on your overall views/listens and customer engagement with content, audio, ads, and combined.

**Heartbeat versus Milestone benefits**

Adobe Analytics for Media is able to be measured through two means: the legacy Milestone method (video only) and the current Heartbeats method (including both the Media SDK and the Media Collection API). The Heartbeats method is the preferred method of measurement and we encourage all clients to move to this version if they haven’t already, to take advantage of the benefits described below.

The legacy Milestone method is based on individual server calls to the Analytics server, for video starts, quartiles, duration, and completes. The Heartbeats method provides a more robust media tracking solution that measures main content in 10-second intervals to provide enhanced, standardized metrics. In addition, Adobe has derived learnings from our Milestone method to provide a smoother, streamlined implementation process through either the Media SDK or Media Collection API utilized by Heartbeats.

Some of the many benefits of the Heartbeats method include:

• **Streamlined implementation process** - Map variables more easily through your player API and validate implementations through the Adobe Debug Tool to ensure all necessary variables are tracked accurately.

• **Automatic Adobe Experience Cloud Integration** - Take advantage of the automatic integration with the Adobe Experience Cloud through the Experience Cloud ID, segment your media audiences, target them, and make media recommendations based on user preferences.

• **Shared data through Federated Analytics** - Capitalize on our industry-first media sharing capabilities, to evaluate data holistically across all of your media distribution partners—operators, programmers, and distributors.

• **Partnerships with Certified Ratings Partners** - Adobe partners with audience ratings partner Nielsen to provide neutral census third party measurement to allow for trusted, certified ratings.

• **Standardized solution across all platforms** - Enable consistent, standardized variables across all of your media and platforms to allow for a more efficient cross-campaign, device and vendor comparison.

**Table 1: Comparison Chart**

<table>
<thead>
<tr>
<th></th>
<th>Media Analytics - Milestone</th>
<th>Media Analytics - Heartbeats</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Media Events</strong></td>
<td>High-level Standard Events</td>
<td>Detailed and Custom Events every 10s for main content, every 1s for ads</td>
</tr>
<tr>
<td><strong>Metrics and Dimensions</strong></td>
<td>Variances among Vendors, Non-Standardized Metrics and Dimensions</td>
<td>Clear, Standardized Metrics, Dimensions, and Benchmarks across Vendors</td>
</tr>
</tbody>
</table>
Adobe Analytics for Media has evolved with the industry to provide strong data collection tools to ensure each media stream is collected and reported across all meaningful devices. Our Media SDK is developed for all of the most utilized devices, including:

- iOS and Android smartphones and tablets
- OTT devices for ROKU, AppleTV, FireTV, and Android TV
- JavaScript Browser for Desktop and Laptop

The SDKs are routinely updated when new versions of devices are released, and you can use these SDKs to integrate with most of the largest media players today, including Brightcove and Ooyala.

For devices or platforms that do not currently have SDK support (or even if they do), you can implement the Media Collection API, through which you make RESTful API calls directly from the device/platform to the Media Analytics backend.

The table below provides a list of the devices that are currently supported through our Media SDK implementation and Media Collection API implementation. To download the most recent version of the SDK, see Download SDKs. If there is a device that is not listed which you are seeking measurement against, please contact customer care or your solution consultant for the status of that device.

<table>
<thead>
<tr>
<th>Devices supported</th>
<th>Media SDK</th>
<th>Media Collection API</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript Browser</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>iOS Devices</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Android Devices</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Unified Windows Platforms (UWP)</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Blackberry</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Apple TV (new/legacy)</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>ROKU (JS)</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>ROKU (Native app)</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>OSX</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
For Media SDK, also see Minimum Platform Version Support

Transport Layer Security (TLS)

TLS Notice -- Adobe has security compliance standards that require the end-of-life of older security protocols. To continue to meet the evolving security protocol standards, Adobe is moving toward the use of TLS 1.2, in order to have the most up-to-date and secure version in use. From February 20th, 2019, Adobe will support only TLS 1.1 or later. With this change, Adobe will no longer collect data from end users with older devices or web browsers deploying TLS 1.0. Migrating to TLS 1.2 provides improved security. It is important that you go through the specifics and plan out the changes for a smooth transition.

💡 Note: TLS is currently the most-widely deployed security protocol used in web browsers and other applications that require data to be securely exchanged over a network.
Measurement options

Media Module Milestone tracking

Milestone overview

: This measurement option has been deprecated.

Legacy Milestone documentation

Configuration

Milestone Video Configuration

To track video, designate a set of Custom Conversion Variables (eVars) and Custom Events for use in tracking and reporting. One Custom Insight variable (s.prop) is also used for pathing.

The variables you select for each metric are added to the video configuration page. This lets the system automatically generate and format the standard video reports. The video name eVar and the video views counter are both required. Other variables are optional but recommended for complete measurement. After video tracking is enabled, you can view reports generated from video data you have reported using video tracking.

You can also track any number of additional metrics for video. For example, if you use multiple video players on your site, you might populate an eVar with the player name. Some of the variables you select might also be used in other areas of your site. For example, if used across your site, the content type variable can let you measure what percentage of your page views are coming from video, and let you relate conversion events to video.

Milestone Reporting Configuration - To set-up video reporting for a Milestone implementation, go to Admin > Report Suite Manager. Select the report suite, then choose Video Management > Video Reporting:

On the first screen, only Video Core will work with Milestone data. Select Video Core and click Save.

On the next screen, select Use Custom Variables.
On the final screen, select the two eVars and three events to be used with your video measurement:

### Video variable reference

The following table contains additional details on the commerce variables and custom events for video:

<table>
<thead>
<tr>
<th>Video Metric</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>eVar</td>
<td>(Required) Collects the name of the video, as specified in the implementation.</td>
</tr>
<tr>
<td></td>
<td>Default expiration: Visit</td>
<td></td>
</tr>
<tr>
<td>Content Type</td>
<td>eVar</td>
<td>Collects data about the type of content viewed by a visitor. Hits sent by video measurement are assigned a content type of <strong>video</strong>.</td>
</tr>
<tr>
<td></td>
<td>Default expiration: Page view</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This variable does not need to be reserved exclusively for video tracking. Having other content report content type using this same variable lets you analyze the distribution of visitors across the different types of content. For example, you could tag other content types using values such as article or product page using this variable.</td>
</tr>
<tr>
<td>Video Metric</td>
<td>Variable Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>From a video measurement perspective, <em>Content Type</em> lets you identify video visitors and thereby calculate video conversion rates.</td>
<td></td>
</tr>
<tr>
<td>Content Time Spent</td>
<td>Event</td>
<td>Counts the time, in seconds, spent watching a video since the last data collection process (image request).</td>
</tr>
<tr>
<td>Video Initiates</td>
<td>Event</td>
<td>Indicates that a visitor has viewed some portion of a video. However, it does not provide any information about how much, or what part, of a video the visitor viewed.</td>
</tr>
<tr>
<td>Video Completes</td>
<td>Event</td>
<td>Indicates that a user has viewed a complete video. By default, the complete event is measured 1 second before the end of the video. During implementation, you can specify how many seconds from the end of the video you would like to consider a view complete. For live video and other streams that don’t have a defined end, you can specify a custom point to measure completes. For example, after a specific time viewed.</td>
</tr>
</tbody>
</table>

**Media Module variables**

The following variables let you configure video measurement. You must define values for the variables in the Required Variables table. Additionally, to track events in your video player, you must enable autoTrack (for supported players) or implement custom player event tracking using the open, play, stop, and close methods.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| Media.trackUsingContextData  | **Syntax:**  
  s.Media.trackUsingContextData = true;  
  This option enables integrated video tracking. When set to true, the media module generates context data for media tracking, instead of the legacy `pev3` value used in previous versions of video measurement.  
  Use `Media.contextDataMapping` to map the context data to the selected eVars and Events.  
  **Default value:** `false` |
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Media.contextDataMapping</code></td>
<td><strong>Syntax:</strong> <code>s.Media.contextDataMapping = {</code>&lt;br&gt;<code>   &quot;a.media.name&quot;:&quot;eVar2,prop2&quot;,&quot;&lt;br&gt;</code>&lt;br&gt;<code>   &quot;a.media.segment&quot;:&quot;eVar3&quot;,&quot;&lt;br&gt;</code>&lt;br&gt;<code>   &quot;a.contentType&quot;:&quot;eVar1&quot;,&quot;&lt;br&gt;</code>&lt;br&gt;<code>   &quot;a.media.timePlayed&quot;:&quot;event3&quot;,&quot;&lt;br&gt;</code>&lt;br&gt;<code>   &quot;a.media.view&quot;:&quot;event1&quot;,&quot;&lt;br&gt;</code>&lt;br&gt;<code>   &quot;a.media.segmentView&quot;:&quot;event2&quot;,&quot;&lt;br&gt;</code>&lt;br&gt;<code>   &quot;a.media.complete&quot;:&quot;event7&quot;,&quot;&lt;br&gt;</code>&lt;br&gt;<code>   &quot;a.media.milestones&quot;:{&lt;br&gt;</code> <code>25&quot;:&quot;event4&quot;,&quot;&lt;br&gt;</code> <code>50&quot;:&quot;event5&quot;,&quot;&lt;br&gt;</code> <code>75&quot;:&quot;event6&quot;&lt;br&gt;</code> })`&lt;br&gt;</td>
</tr>
</tbody>
</table>
|                                | An object that defines variable mapping to eVars and Events that you want to use for video measurement. The object must map the following fields:<br><br>`a.media.name:` (Required) Populates variables with the video name. Provide the eVar that you selected to store the video name, and the *Custom Insight Video* variable (s.prop) you want to use for video pathing. Provide the values in a comma-separated list.<br><br>`a.media.segment:` (Optional) The eVar that you want to store the media segment name.<br><br>`a.contentType:` (Optional) The eVar that you want to store the video value, which contains visit and visitor tracking enabled to generate video visit and visitor reporting. The variable you select is likely already used to store data such as article slide show or product page.<br><br>`a.media.view:` (Required) The Event that you want to count media views.<br><br>`a.media.segmentView:` (Optional) The Event that you want to count segment views.<br><br>`a.media.complete:` (Optional) The Event that you want to count complete views.<br><br>`a.media.timePlayed:` (Optional, highly recommended) The numeric Event that you want to store the number of video seconds played.<br><br>`a.media.milestones:` (Optional) An object that maps s.Media.trackMilestones milestones to counter Events. Media.segmentByMilestones should be set to true if you define milestones.<br><br>**Ad tracking**<br>To track ads, the following context data variables are available:<br><br>`a.media.ad.name:` (Required) Populates variables with the ad name. Provide the eVar that you selected to store the ad name, and the *Custom Insight Video* variable (s.prop) you want to use for pathing. Provide the values in a comma-separated list.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.media.ad.pod</td>
<td>The position in the primary content the ad was played.</td>
</tr>
<tr>
<td>a.media.ad.podPosition</td>
<td>The position within the pod where the ad is played.</td>
</tr>
<tr>
<td>a.media.ad.CPM</td>
<td>The CPM or encrypted CPM (prefixed with a &quot;~&quot;) that applies to this playback.</td>
</tr>
<tr>
<td>a.media.ad.view</td>
<td>Works the same as a.media.view.</td>
</tr>
<tr>
<td>a.media.ad.clicked</td>
<td>Count the number of clicks for the ad (Media.click calls).</td>
</tr>
<tr>
<td>a.media.ad.timePlayed</td>
<td>Works the same as a.media.timePlayed.</td>
</tr>
<tr>
<td>a.media.ad.complete</td>
<td>Works the same as a.media.complete.</td>
</tr>
<tr>
<td>a.media.ad.segment</td>
<td>Works the same as a.media.segment.</td>
</tr>
<tr>
<td>a.media.ad.segmentView</td>
<td>Works the same as a.media.segmentView.</td>
</tr>
<tr>
<td>a.media.ad.milestones</td>
<td>Works the same as a.media.milestones.</td>
</tr>
<tr>
<td>a.media.ad.offsetMilestones</td>
<td>Works the same as a.media.offsetMilestones.</td>
</tr>
</tbody>
</table>

**Media.trackVars**

**Syntax:**

```
s.Media.trackVars = "events,prop2,eVar1,eVar2,eVar3";
```

A comma-separated list of all variables that are set in your video tracking code.

**Media.trackEvents**

**Syntax:**

```
s.Media.trackEvents = "event1,event2,event3,event4,event5,event6,event7"
```

A comma-separated list of all events that are set in your video tracking code.

**Optional variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.autoTrack</td>
<td>Enables automatic tracking for supported players. Supported players are as follows:</td>
</tr>
<tr>
<td></td>
<td>• Open Source Media Framework (OSMF)</td>
</tr>
<tr>
<td></td>
<td>• FLVPlayback (Video players created by the import video wizard in Flash Professional)</td>
</tr>
<tr>
<td></td>
<td>• Silverlight</td>
</tr>
</tbody>
</table>
### Variable Description

- MediaDisplay
- MediaPlayback
- Brightcove API versions 2 & 3 (see Brightcove)
- Windows Media Player, Quicktime, or Real Player using JavaScript

If you are not using one of the above players you can use `Media.open`, `Media.play`, `Media.stop`, and `Media.close` to track player events.

---

**Media.autoTrackNetStreams**

**Syntax:**

```
s.Media.autoTrackNetStreams = true
```

Flash 10.3 introduced new functionality to the NetStream component that enables enhanced video tracking. If you are using a custom Flash NetStream player you can enable this variable to enable functionality similar to autoTrack. This method requires that videos are viewed in Flash 10.3 or later.

---

**Media.completeByCloseOffset**

**Syntax:**

```
s.Media.completeByCloseOffset = true
```

This setting lets you count a complete video view a few seconds before the actual end of the video.

The event is sent based on the number of seconds specified in `completeCloseOffsetThreshold`. This lets you measure completes in video players that never report an offset equal to the length of the video.

By default, this value is set to true and the threshold is set to 1 second. With these defaults the complete event is sent 1 second before the end of the video.

---

**Media.completeCloseOffsetThreshold**

**Syntax:**

```
s.Media.completeCloseOffsetThreshold = 1
```

This threshold lets you count a complete video view a few seconds before the actual end of the video. **Media.completeByCloseOffset** must be set to true to use this threshold.

The integer value you supply determines how far off in seconds the offset can be from the length of the video at close and still count as a complete. This lets you measure completes in video players that never report an offset equal to the length of the video.

The default threshold is 1 second.

---

**Media.playerName**

**Syntax:**

```
s.Media.playerName = "Custom Player Name"
```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.Media.trackSeconds = 15</td>
<td>Defines the interval, in seconds, for sending video tracking data to Adobe data collection servers while the video is playing. The value must be set in increments of 5 seconds. Enabling Media.trackSeconds triggers only the events that are defined in Media.contextDataMapping. To send additional variables outside of those specified for video measurement, you must use Media.Monitor.</td>
</tr>
<tr>
<td>Media.trackMilestones</td>
<td>Tracks milestones as percentage of the video length. <strong>Syntax:</strong> s.Media.trackMilestones = &quot;25,50,75&quot;; Defines the interval, as a percentage of the video length, for sending video tracking data to Adobe data collection servers. Specify the milestones as a comma-separated list of whole numbers. For example: 10 = 10%, 23 = 23%. Because these milestones are fixed points in the video, if a visitor views past the 10% milestone, then rewinds and passes the 10% milestone again, the media module sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, the media module does not send the tracking data for that milestone. Enabling Media.trackMilestones triggers only the events that are defined in Media.contextDataMapping. To send additional variables outside of those specified for video measurement, you must use Media.monitor.</td>
</tr>
<tr>
<td>Media.trackOffsetMilestones</td>
<td>Tracks milestones as seconds elapsed from the beginning of the video. <strong>Syntax:</strong> s.Media.trackOffsetMilestones = &quot;20,40,60&quot;; Defines the interval, as seconds elapsed from the beginning of the video, for sending video tracking data to Adobe data collection servers. Specify the milestones as a comma-separated list of whole numbers. For example: 20 = 20 seconds, 40 = 40 seconds). Because these milestones are fixed points in the video, if a visitor views past the 20 seconds milestone, then rewinds and passes the 20 seconds milestone again, the media module sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, the media module does not send the tracking data for that milestone.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Media.segmentByMilestones                    | Syntax:                                                                                                         
|                                               | s.Media.segmentByMilestones = true;                                                                                                                   |
|                                               | Automatically generates the segment name, segment number, and segment length data, based on the length of the media and the milestones specified in Media.trackMilestones. |
|                                               | Segmenting by milestones is the only way to define segments when using autoTrack.                                                                       |
|                                               | Default value: false.                                                                                                                                             |
| Media.segmentByOffsetMilestones              | Syntax:                                                                                                         
|                                               | s.Media.segmentByOffsetMilestones = true;                                                                                                                   |
|                                               | Automatically generates the segment name, segment number, and segment length data, based on the length of the media and the milestones specified in Media.trackOffsetMilestones. |
|                                               | Segmenting by milestones is the only way to define segments when using autoTrack.                                                                       |
|                                               | Default value: false.                                                                                                                                             |

**Ad Tracking variables**

These variables are used to send ad information in conjunction with the openAd method. See [VAST Video Ad Tracking](#).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.adTrackSeconds</td>
<td>s.Media.adTrackSeconds = 15</td>
</tr>
<tr>
<td></td>
<td>Defines the interval, in seconds, for sending video ad tracking data to Adobe data collection servers while the video is playing. The value must be set in increments of 5 seconds.</td>
</tr>
<tr>
<td></td>
<td>Enabling Media.adTrackSeconds triggers only the events that are defined in Media.contextDataMapping. To send additional variables outside of those specified for video measurement, you must use Media.monitor.</td>
</tr>
<tr>
<td>Media.adTrackMilestones</td>
<td>Tracks ad milestones as percentage of the ad length.</td>
</tr>
</tbody>
</table>
|                                   | Syntax:                                                                                                         
<p>|                                   | s.Media.adTrackMilestones = &quot;25,50,75&quot;;                                                                                                                  |</p>
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.adTrackMilestones</td>
<td>Defines the interval, as a percentage of the ad length, for sending ad tracking data to Adobe data collection servers. Specify the milestones as a comma-separated list of whole numbers. For example: 10 = 10%, 23 = 23%). Because these milestones are fixed points in the ad, if a visitor views past the 10% milestone, then rewinds and passes the 10% milestone again, the media module sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, the media module does not send the tracking data for that milestone. Enabling Media.adTrackMilestones triggers only the events that are defined in Media.contextDataMapping. To send additional variables outside of those specified for video measurement, you must use Media.monitor.</td>
</tr>
</tbody>
</table>
| Media.adTrackOffsetMilestones | Tracks ad milestones as seconds elapsed from the beginning of the ad. Syntax: 

```javascript
s.Media.adTrackOffsetMilestones = "20,40,60";
```

Defines the interval, as seconds elapsed from the beginning of the ad, for sending ad tracking data to Adobe data collection servers. Specify the milestones as a comma-separated list of whole numbers. For example: 20 = 20 seconds, 40 = 40 seconds). Because these milestones are fixed points in the ad, if a visitor views past the 20 seconds milestone, then rewinds and passes the 20 seconds milestone again, the media module sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, the media module does not send the tracking data for that milestone. Enabling Media.adTrackOffsetMilestones triggers only the events that are defined in Media.contextDataMapping. To send additional variables outside of those specified for video measurement, you must use Media.monitor. |
| Media.adSegmentByMilestones | Syntax: 

```javascript
s.Media.adSegmentByMilestones = true;
```

Automatically generates the segment name, segment number, and segment length data, based on the length of the media and the milestones specified in Media.adTrackMilestones. Segmenting by milestones is the only way to define segments when using autoTrack. Default value: false. |
<p>| Media.adSegmentByOffsetMilestones | Syntax: |</p>
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.Media.adSegmentByOffsetMilestones = true;</td>
<td>Automatically generates the segment name, segment number, and segment length data, based on the length of the media and the milestones specified in Media.adTrackOffsetMilestones. Segmenting by milestones is the only way to define segments when using autoTrack. Default value: false.</td>
</tr>
</tbody>
</table>

**Media Module methods**

The media module methods are used to manually tracking player events and to track additional metrics that are not part of the standard video reports.

If you are using Media.autoTrack and are not tracking additional metrics, you do not need to call any of these methods directly. All arguments are required unless specified as optional.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| Media.open | Syntax:  
s.Media.open(mediaName, mediaLength, mediaPlayerName)  
Prepares the media module to collect video tracking data. This method takes the following parameters:  
mediaName: (Required) The name of the video as you want it to appear in video reports.  
mediaLength: (Required) The length of the video in seconds.  
mediaPlayerName: (Required) The name of the media player used to view the video, as you want it to appear in video reports. |
| Media.openAd | Syntax:  
s.Media.openAd(name, length, playerName, parentName, parentPod, parentPodPosition, CPM)  
Prepares the media module to collect ad tracking data. This method takes the following parameters:  
• name: (Required) The name or ID of the ad.  
• length: (Required) The length of the ad.  
• playerName: (Required) The name of the media player used to view the ad.  
• parentName: The name or ID of the primary content where the ad is embedded.  
• parentPod: The position in the primary content the ad was played.  
• parentPodPosition: The position within the pod where the ad is played. |
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>* CPM: CPM or encrypted CPM (prefixed with a &quot;~&quot;) that applies to this playback.</td>
<td></td>
</tr>
<tr>
<td>Media.click</td>
<td><strong>Syntax:</strong> ( s.Media.click(name, offset) )  ( s.Media.click(name, offset) )</td>
</tr>
<tr>
<td></td>
<td>Track when an ad is clicked in a video. This method takes the following parameters: ( name: ) The name of the ad. This must match the name used in Media.openAd. ( offset: ) The offset into the ad when the click occurred.</td>
</tr>
<tr>
<td>Media.close</td>
<td><strong>Syntax:</strong> ( s.Media.close(mediaName) )</td>
</tr>
<tr>
<td></td>
<td>Ends video data collection and sends information to Adobe data collection servers. Call this method at the end of the video. This method takes the following parameter: ( mediaName: ) The name of the video. This must match the name used in Media.open.</td>
</tr>
<tr>
<td>Media.complete</td>
<td><strong>Syntax:</strong> ( s.Media.complete(name, offset) )</td>
</tr>
<tr>
<td></td>
<td>This method manually tracks a complete event. This method is used when you need to trigger events using special logic that can't be handled using Media.completeByCloseOffset.</td>
</tr>
<tr>
<td></td>
<td>For example, if you are measuring a live stream that has no defined end, you might trigger a complete after a user views a live stream for ( X ) seconds. You might measure a complete using a percentage calculation based on the length and type of content. This method takes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>( mediaName: ) The name of the video. This must match the name used in Media.open. ( mediaOffset: ) The number of seconds into the video when the complete event should be sent. Specify the offset based on the video starting at second zero. If your media player tracks using milliseconds, make sure the value is converted to seconds before you call Media.complete.</td>
</tr>
<tr>
<td></td>
<td>If you plan to call complete manually, set ( s.Media.completeByCloseOffset = false ) to disable automatic triggering of the complete event.</td>
</tr>
<tr>
<td>Media.play</td>
<td><strong>Syntax:</strong></td>
</tr>
</tbody>
</table>
### Method

s.Media.play(name, offset, segmentNum, segment, segmentLength)

Call this method anytime a video starts playing. When using manual video measurement, you can provide the current segment data when sending video measurement data.

If your player changes from one segment to another, for whatever reason, you should call `Media.stop` before calling `Media.play` again for the new segment.

This method takes the following parameters:

- **mediaName**: The name of the video. This must match the name used in `Media.open`.
- **mediaOffset**: The number of seconds into the video that play begins. Specify the offset based on the video starting at second zero. If your media player tracks using milliseconds, make sure the value is converted to seconds before you call `Media.play`.
- **segmentNum**: (Optional) The current segment number, which marketing reports use to order the display of segments in reports. The `segmentNum` parameter must be greater than zero.
- **segment**: (Optional) The current segment name.
- **segmentLength**: (Optional) The current segment length, in seconds.

For example:

```javascript
s.Media.play("My Video", 1800, 2, "Second Quarter", 1800)
s.Media.play("My Video", 0, 1, "Preroll", 30)
```

### Media.stop

**Syntax:**

```javascript
s.Media.stop(mediaName, mediaOffset)
```

Tracks a stop event (stop, pause, etc.) for the specified video. This method takes the following parameters:

- **mediaName**: The name of the video. This must match the name used in `Media.open`.
- **mediaOffset**: The number of seconds into the video that the stop or pause event occurs. Specify the offset based on the video starting at second zero.

### Media.monitor

**Syntax:**

```javascript
s.Media.monitor(s, media)
```

**Silverlight Syntax:**

```javascript
s.Media.monitor = new AppMeasurement_Media_Monitor(myMediaMonitor);
```
The Silverlight app media monitor implements the Objective-C delegate design pattern. `myMediaMonitor` is a class method that takes the `s` and `media` parameters.

Use this method to send additional video metrics. You can setup additional variables (Props, eVars, Events) and send them using `Media.track` based on the current state of the video as it is playing.

See *Measuring Additional Metrics using Media.monitor*.

This method takes the following parameters:

- `s`: The `AppMeasurement` instance (or JavaScript `s` object).
- `media`: An object with members providing the state of the video. These members include:
  - `media.name`: The name of the video. This must match the name used in `Media.open`.
  - `media.length`: The length of the video in seconds given in the call to `Media.open`.
  - `media.playerName`: The name of the media player given in the call to `Media.open`.
  - `media.mediaEvent`: A string containing the event name that caused the monitor call. These events are:
    - `OPEN`: When playback is first observed through `Media.autoTrack` or a call to `Media.play`.
    - `CLOSE`: When playback ends at the completion of the video through `Media.autoTrack` or a call to `Media.close`.
    - `PLAY`: When playback resumes after being paused or scrubbing through `Media.autoTrack` or a second call to `Media.play`.
    - `STOP`: When playback stops due to a pause of the beginning of scrubbing through `Media.autoTrack` or a call to `Media.stop`.
    - `MONITOR`: When our automatic monitoring checks the state of the video while it's playing (every second).
    - `SECONDS`: At the second interval defined by the `Media.trackSeconds` variable.
    - `MILESTONE`: At the milestones defined by the `Media.trackMilestones` variable.
  - `media.openTime`: An `NSDate` object containing data about when `Media.open` was called.
  - `media.offset`: The current offset, in seconds, (actual point in the video) into the video. The offset starts at zero (the first second of the video is second 0).
### Method Description

- **media.percent**: The current percentage of the video that has played, based on the video length and the current offset.
- **media.timePlayed**: The total number of seconds played so far.
- **media.eventFirstTime**: Indicates if this was the first time this media event was called for this video.

### Syntax

`Media.track

s.Media.track(mediaName)`

Immediately sends the current video state, along with any `Media.trackVars` and `Media.trackEvents` you’ve defined. This method is used within `Media.monitor`.

See [Measuring Additional Metrics using Media.monitor](#).

Call `Media.open` and `Media.play` on the video before calling this method. This method takes the following parameter:

- **mediaName**: The name of the video. This must match the name used in `Media.open`.

This method is the only way to send additional variables while the video is playing. It resets the seconds interval and percent milestone counters to zero to prevent multiple tracking hits.

### Track video player events

You can track media players by creating functions attached to the video player event handlers. This lets you call `Media.open`, `Media.play`, `Media.stop`, and `Media.close` at the appropriate times. For example:

- **Load**: Call `Media.open` and `Media.play`
- **Pause**: Call `Media.stop`. For example, if a user pauses a video after 15 seconds, call `s.Media.stop("Video1",15)`
- **Buffer**: Call `Media.stop` while the video buffers. Call `Media.play` when playback resumes.
- **Resume**: Call `Media.play`. For example, when a user resumes a video after initially playing 15 seconds of the video, call `s.Media.play("Video1",15)`.
- **Scrub (slider)**: When the user drags the video slider, call `Media.stop`. When the user releases the video slider, call `Media.play`.
- **End**: Call `Media.stop`, then `Media.close`. For example, at the end of a 100-second video, call `s.Media.stop("Video1",100), then s.Media.close("Video1")`.

To accomplish this, you can define four custom functions that you can call from the media player event handlers. The various parameters passed into `Media.open`, `Media.play`, `Media.stop`, and `Media.close` come from the player. The following pseudocode demonstrates how this might be done:

```javascript
/* Call on video load */
function startMovie() {
    s.Media.open(mediaName,mediaLength,mediaPlayerName);
```
playMovie();
}

/* Call on video resume from pause and slider release */
function playMovie() {
    s.Media.play(mediaName,
            mediaOffset,
            segmentNum,
            segment,
            segmentLength);
}

/* Call on video pause and slider grab */
function stopMovie() {
    s.Media.stop(mediaName,mediaOffset);
}

/* Call on video end */
/* Measuring Video for Developers 43 */
function endMovie() {
    stopMovie();
    s.Media.close(mediaName);
}

JavaScript autotrack
The JavaScript media module identifies all `<embed>` or `<object>` tags in the page HTML. It then searches the data in each tag to determine which media player, if any, is being used. If the player is Windows Media Player, Quicktime, or Real Player, autotrack can be used, though autotrack for Windows media player works only with Internet Explorer. Manual tracking for Windows Media Player is required to support all other browsers.

You must have the `classid` attribute set on the object you want to track. The `classid` is required to expose the event handlers used by the Media Module to automatically track the video.

s.Media.autotrack = true

JavaScript sample code

// Sample implementation
s.usePlugins=true
function s_doPlugins(s) {
    /* Add manual calls to modules and plugins here */
}

s.doPlugins=s_doPlugins

/**********Media Module Calls***********/
s.loadModule("Media")

/*Configure Media Module Functions */
s.Media.autoTrack= true;
s.Media.trackVars="events,prop2,eVar1,eVar2,eVar3";
s.Media.trackEvents="event1,event2,event3,event4,event5,event6,event7"
s.Media.trackMilestones="25,50,75";
s.Media.mediaName="My Media Player";
s.Media.segmentByMilestones = true;
s.Media.trackUsingContextData = true;
s.Media.contextDataMapping = {
    "a.media.name":"eVar2,prop2",
    "a.media.segment":"eVar3",
    "a.contentType":"eVar1",
    "a.media.timePlayed":"event3",
    "a.media.view":"event1",
    "a.media.segmentView":"event2",
    "a.media.complete":"event7",
    "a.media.milestones":{
        25:"event4",
        50:"event5",
        75:"event6"
    }
}
Migrating from Milestone to Media Analytics

Overview
The core concepts of video measurement are the same for Milestone and Media Analytics, which is taking video player events and mapping them to analytics methods, while also grabbing player metadata and values and mapping them to analytics variables. The Media Analytics solution grew out of Milestone, so many of the methods and metrics are the same, however, the configuration approach and code has changed significantly. It should be possible to update the player event code to point to the new Media Analytics methods. See Set up and configure and Track core for more details on implementing Media Analytics.

The following tables provide translations between the Milestone solution and the Media Analytics solution.

**Migration guide**

**Variable Reference:**

<table>
<thead>
<tr>
<th>Milestone Metric</th>
<th>Variable Type</th>
<th>Media Analytics Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>eVar</td>
<td>Content</td>
</tr>
<tr>
<td></td>
<td>Default expiration: Visit</td>
<td></td>
</tr>
<tr>
<td>Content Type</td>
<td>eVar</td>
<td>Content Type</td>
</tr>
<tr>
<td></td>
<td>Default expiration: Page view</td>
<td></td>
</tr>
<tr>
<td>Content Time Spent</td>
<td>Event</td>
<td>Content Time Spent</td>
</tr>
<tr>
<td></td>
<td>Type: Counter</td>
<td></td>
</tr>
<tr>
<td>Milestone Metric</td>
<td>Variable Type</td>
<td>Media Analytics Metric</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Video Initiates</td>
<td>Event</td>
<td>Video Initiates</td>
</tr>
<tr>
<td></td>
<td>Type: Counter</td>
<td></td>
</tr>
<tr>
<td>Video Completes</td>
<td>Event</td>
<td>Content Complete</td>
</tr>
<tr>
<td></td>
<td>Type: Counter</td>
<td></td>
</tr>
</tbody>
</table>

**Media Module variables:**

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Milestone Syntax</th>
<th>Media Analytics</th>
<th>Media Analytics Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.trackUsingContextData</td>
<td>s.Media.trackUsingContextData = true;</td>
<td>N/A</td>
<td>All Media Analytics data is only sent using Context Data.</td>
</tr>
</tbody>
</table>
| Media.contextDataMapping | s.Media.contextDataMapping = {  
|                        | "a.media.name":"eVar2,prop2", 
|                        | "a.media.segment":"eVar3",  
|                        | "a.contentType":"eVar1", 
|                        | "a.media.timePlayed":"event3", 
|                        | "a.media.view":"event1", 
|                        | "a.media.segmentView":"event2", 
|                        | "a.media.complete":"event7", 
|                        | "a.media.milestones": {  
|                        | 25:"event4", 
|                        | 50:"event5", 
|                        | 75:"event6"  
|                        | }  
|                        | };                                                                              | N/A             | Media Analytics context data is automatically populated into reserved variables. Mapping to eVars, props, and events I no longer needed within the implementation code. Customers can map context data to variables using processing rules. |
| Media.trackVars      | s.Media.trackVars = "events, prop2, eVar1, eVar2, eVar3";                      | N/A             | No longer needed since mapping happens via reserved variables and processing rules.     |
| Media.trackEvents    | s.Media.trackEvents = "event1, event2, event3, event4, event5, event6, event7" | N/A             | No longer needed since mapping happens via reserved variables and processing rules.     |

Optional variables
<table>
<thead>
<tr>
<th>Milestone</th>
<th>Milestone Syntax</th>
<th>Media Analytics</th>
<th>Media Analytics Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.autoTrack</td>
<td>s.Media.autoTrack = true;</td>
<td>N/A</td>
<td>We no longer provide pre-built player mappings.</td>
</tr>
<tr>
<td>MediaTrackNetStreams</td>
<td>s.Media.autoTrackNetStreams = true</td>
<td>N/A</td>
<td>We no longer provide pre-built player mappings.</td>
</tr>
<tr>
<td>Media.completeByCloseOffset</td>
<td>s.Media.completeByCloseOffset = true</td>
<td>N/A</td>
<td>Content Complete only supports a 100% progress marker.</td>
</tr>
<tr>
<td>Media.completeCloseOffsetThreshold</td>
<td>s.Media.completeCloseOffsetThreshold = 1</td>
<td>N/A</td>
<td>Content Complete only supports a 100% progress marker.</td>
</tr>
<tr>
<td>Media.playerName</td>
<td>s.Media.playerName = &quot;Custom Player Name&quot;</td>
<td></td>
<td>MediaAnalyticsConfig.playerName</td>
</tr>
<tr>
<td>Media.trackSeconds</td>
<td>s.Media.trackSeconds = 15</td>
<td>N/A</td>
<td>Media Analytics is set to 10 seconds for content and 1 second for ads. No other options are available.</td>
</tr>
<tr>
<td>Media.trackMilestones</td>
<td>s.Media.trackMilestones = &quot;25,50,75&quot;;</td>
<td>N/A</td>
<td>Media Analytics always tracks progress markers at 10%, 25%, 50%, 75%, 95%</td>
</tr>
<tr>
<td>Media.trackOffsetMilestones</td>
<td>s.Media.trackOffsetMilestones = &quot;20,40,60&quot;;</td>
<td>N/A</td>
<td>Media Analytics always tracks progress markers at 10%, 25%, 50%, 75%, 95%</td>
</tr>
<tr>
<td>Media.segmentByMilestones</td>
<td>s.Media.segmentByMilestones = true;</td>
<td>N/A</td>
<td>Auto track is no longer available</td>
</tr>
<tr>
<td>Media.segmentByOffsetMilestones</td>
<td>s.Media.segmentByOffsetMilestones = true;</td>
<td>N/A</td>
<td>Auto track is no longer available</td>
</tr>
</tbody>
</table>

Ad tracking variables:
<table>
<thead>
<tr>
<th>Milestone</th>
<th>Milestone Syntax</th>
<th>Media Analytics</th>
<th>Media Analytics Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.adTrackSeconds</td>
<td>s.Media.adTrackSeconds</td>
<td>N/A</td>
<td>Media Analytics is set to 10 seconds for content and 1 second for ads. No other options are available.</td>
</tr>
<tr>
<td>Media.adTrackMilestones</td>
<td>s.Media.adTrackMilestones = &quot;25,50,75&quot;;</td>
<td>N/A</td>
<td>Progress markers are not provided by default for ads. Use calculated metrics to build ad progress markers.</td>
</tr>
<tr>
<td>Media.adTrackOffsetMilestones</td>
<td>s.Media.adTrackOffsetMilestones = &quot;20,40,60&quot;;</td>
<td>N/A</td>
<td>Media Analytics is set to 1 second for ads. No other options are available.</td>
</tr>
<tr>
<td>Media.adSegmentByMilestones</td>
<td>s.Media.adSegmentByMilestones = true;</td>
<td>N/A</td>
<td>Auto track is no longer available</td>
</tr>
<tr>
<td>Media.adSegmentByOffsetMilestones</td>
<td>s.Media.adSegmentByOffsetMilestones = true;</td>
<td>N/A</td>
<td>Auto track is no longer available</td>
</tr>
</tbody>
</table>

Media Module methods:
<table>
<thead>
<tr>
<th>Milestone</th>
<th>Milestone Syntax</th>
<th>Media Analytics</th>
<th>Media Analytics Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.open</td>
<td>s.Media.open(mediaName,</td>
<td>trackSessionStart</td>
<td>trackSessionStart(mediaObject, contextData)</td>
</tr>
<tr>
<td></td>
<td>mediaLength,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mediaPlayerName)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediaName</td>
<td>mediaName:(Required) The name of the video as you want it to appear in video reports.</td>
<td>name</td>
<td>createMediaObject(name, mediaId, length, streamType)</td>
</tr>
<tr>
<td>mediaLength</td>
<td>mediaLength:(Required) The length of the video in seconds.</td>
<td>length</td>
<td>createMediaObject(name, mediaId, length, streamType)</td>
</tr>
<tr>
<td>mediaPlayerName</td>
<td>mediaPlayerName: (Required) The name of the media player used to view the video, as you want it to appear in video reports.</td>
<td>playerName</td>
<td>MediaHeartbeatConfig.playerName</td>
</tr>
<tr>
<td>Media.openAd</td>
<td>s.Media.openAd(name,</td>
<td>trackEvent</td>
<td>mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakObject);</td>
</tr>
<tr>
<td></td>
<td>length,</td>
<td>MediaHeartbeat.Event.AdBreakStart,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>playerName,</td>
<td>adObject,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parentName,</td>
<td>adCustomMetadata);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parentPod,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>parentPodPosition,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>name:(Required) The name or ID of the ad.</td>
<td>name</td>
<td>createAdObject(name, adId, position, length)</td>
</tr>
<tr>
<td>length</td>
<td>length:(Required) The length of the ad.</td>
<td>length</td>
<td>createAdObject(name, adId, position, length)</td>
</tr>
<tr>
<td>playerName</td>
<td>playerName:(Required) The name of the media player used to view the ad.</td>
<td>playerName</td>
<td>MediaHeartbeatConfig.playerName</td>
</tr>
<tr>
<td>Milestone</td>
<td>Milestone Syntax</td>
<td>Media Analytics</td>
<td>Media Analytics Syntax</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>parentName</td>
<td><code>parentName</code>: The name or ID of the primary content where the ad is embedded.</td>
<td>N/A</td>
<td>Automatically inherited</td>
</tr>
<tr>
<td>parentPod</td>
<td><code>parentPod</code>: The position in the primary content the ad was played.</td>
<td>position</td>
<td><code>createAdBreakObject(name, position, startTime)</code></td>
</tr>
<tr>
<td>parentPodPosition</td>
<td><code>parentPodPosition</code>: The position within the pod where the ad is played.</td>
<td>position</td>
<td><code>createAdObject(name, adId, position, length)</code></td>
</tr>
<tr>
<td>CPM</td>
<td>CPM: The CPM or encrypted CPM (prefixed with a &quot;~&quot;) that applies to this playback.</td>
<td>N/A</td>
<td>Not available by default in Media Analytics</td>
</tr>
<tr>
<td>Milestone</td>
<td>Milestone Syntax</td>
<td>Media Analytics</td>
<td>Media Analytics Syntax</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------</td>
<td>----------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Media.click</td>
<td>s.Media.click(name,offset)</td>
<td>N/A</td>
<td>Use a custom link analytics call to track clicks</td>
</tr>
<tr>
<td>Media.close</td>
<td>s.Media.close(mediaName)</td>
<td>trackSessionEnd</td>
<td>trackSessionEnd()</td>
</tr>
<tr>
<td>Media.complete</td>
<td>s.Media.complete(name,offset)</td>
<td>trackComplete</td>
<td>trackComplete()</td>
</tr>
<tr>
<td>Media.play</td>
<td>s.Media.play(name,offset,segmentNum,segment,segmentLength)</td>
<td>trackPlay</td>
<td>trackPlay()</td>
</tr>
<tr>
<td>Media.stop</td>
<td>s.Media.stop(mediaName,mediaOffset)</td>
<td>trackPause Or trackEvent</td>
<td>trackPause() Or trackEvent (MediaHeartbeat.Event.SeekStart) Or trackEvent (MediaHeartbeat.Event.BufferStart);</td>
</tr>
<tr>
<td>Media.track</td>
<td>s.Media.track(mediaName)</td>
<td>N/A</td>
<td>Tracking call frequency is automatically set.</td>
</tr>
</tbody>
</table>

**Migrating from Milestone to Custom Link**

**Overview**

The core concepts of video measurement are the same for Milestone and Custom Link tracking, which is taking video player events and mapping them to analytics methods, while also grabbing player metadata and values and mapping them to analytics variables. The Custom Link approach should be considered as a slimming down and simplifying of both the implementation and the data collected. With the Custom Link solution, no variables or methods are pre-defined for video measurement, it requires a full custom set-up. It should be possible to update the player...
event code to point to the custom link tracking calls for basic player events such as start and complete. See *Custom Link implementation guide* and *Manual Link Tracking Using Custom Link Code* for more details.

The following tables provide translations between the Milestone solution and the Custom Link solution.

**Migration guide**

**Video Variable Reference:**

<table>
<thead>
<tr>
<th>Milestone Metric</th>
<th>Variable Type</th>
<th>Custom Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>eVar</td>
<td>Define your own eVar</td>
</tr>
<tr>
<td></td>
<td>Default expiration: Visit</td>
<td></td>
</tr>
<tr>
<td>Content Type</td>
<td>eVar</td>
<td>Define your own eVar</td>
</tr>
<tr>
<td></td>
<td>Default expiration: Page view</td>
<td></td>
</tr>
<tr>
<td>Content Time Spent</td>
<td>Event</td>
<td>Define your own event</td>
</tr>
<tr>
<td></td>
<td>Type: Counter</td>
<td></td>
</tr>
<tr>
<td>Video Initiates</td>
<td>Event</td>
<td>Define your own event</td>
</tr>
<tr>
<td></td>
<td>Type: Counter</td>
<td></td>
</tr>
<tr>
<td>Video Completes</td>
<td>Event</td>
<td>Define your own event</td>
</tr>
<tr>
<td></td>
<td>Type: Counter</td>
<td></td>
</tr>
</tbody>
</table>

**Media Module variables:**
<table>
<thead>
<tr>
<th>Milestone</th>
<th>Milestone Syntax</th>
<th>Custom Link</th>
<th>Custom Link Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.trackUsingContextData</td>
<td>s.Media.trackUsingContextData = true;</td>
<td>s.linkTrackVars = mediaName;</td>
<td>s.linkTrackVars = 'events,'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s.contextData['variable'] = mediaName;</td>
<td>contextData.video.name,'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s.contextData['video.name'] = mediaName;</td>
</tr>
<tr>
<td>Media.contextDataMapping</td>
<td>s.Media.contextDataMapping = {</td>
<td>N/A</td>
<td>Mapping context data to eVars, props, and events is now completed through</td>
</tr>
<tr>
<td></td>
<td>&quot;a.media.name&quot;:&quot;eVar2,prop2&quot;,</td>
<td></td>
<td>processing rules.</td>
</tr>
<tr>
<td></td>
<td>&quot;a.media.segment&quot;:&quot;eVar3&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;a.contentType&quot;:&quot;eVar1&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;a.media.timePlayed&quot;:&quot;event3&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;a.media.view&quot;:&quot;event1&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;a.media.segmentView&quot;:&quot;event2&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;a.media.complete&quot;:&quot;event7&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;a.media.milestones&quot;:{</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>25:&quot;event4&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50:&quot;event5&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75:&quot;event6&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Media.trackVars</td>
<td>s.Media.trackVars = &quot;events, prop2, eVar1, eVar2, eVar3&quot;;</td>
<td>linkTrackVars</td>
<td>s.linkTrackVars = 'events, prop10, eVar10, eVar12, eVar13, eVar15,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>contextData.video.name,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>contextData.video.view';</td>
</tr>
<tr>
<td>Media.trackEvents</td>
<td>s.Media.trackEvents = &quot;event1, event2, event3, event4, event5, event6, event7&quot;</td>
<td>linkTrackEvents</td>
<td>s.linkTrackEvents = 'event2';</td>
</tr>
</tbody>
</table>
### Optional variables:

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Milestone Syntax</th>
<th>Custom Link</th>
<th>Custom Link Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.autoTrack</td>
<td>s.Media.autoTrack = true;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.autoTrackNetStreams</td>
<td>s.Media.autoTrackNetStreams = true</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.completeByCloseOffset</td>
<td>s.Media.completeByCloseOffset = true</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.completeCloseOffsetThreshold</td>
<td>s.Media.completeCloseOffsetThreshold = 1</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.playerName</td>
<td>s.Media.playerName = &quot;Custom Player Name&quot;</td>
<td>Set eVar or context data variable in link call</td>
<td>s.contextData['video.player'] = &quot;CustomPlayer Name&quot;;</td>
</tr>
<tr>
<td>Media.trackSeconds</td>
<td>s.Media.trackSeconds = 15</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.trackMilestones</td>
<td>s.Media.trackMilestones = &quot;25,50,75&quot;;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.trackOffsetMilestones</td>
<td>s.Media.trackOffsetMilestones = &quot;20,40,60&quot;;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.segmentByMilestones</td>
<td>s.Media.segmentByMilestones = true;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.segmentByOffsetMilestones</td>
<td>s.Media.segmentByOffsetMilestones = true;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
</tbody>
</table>

### Ad tracking variables:
### Milestone Syntax

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Milestone Syntax</th>
<th>Custom Link</th>
<th>Custom Link Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.adTrackSeconds</td>
<td>s.Media.adTrackSeconds = 15</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.adTrackMilestones</td>
<td>s.Media.adTrackMilestones = &quot;25,50,75&quot;;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.adTrackOffsetMilestones</td>
<td>s.Media.adTrackOffsetMilestones = &quot;20,40,60&quot;;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.adSegmentByMilestones</td>
<td>s.Media.adSegmentByMilestones = true;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.adSegmentByOffsetMilestones</td>
<td>s.Media.adSegmentByOffsetMilestones = true;</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
</tbody>
</table>

### Media Module methods:

- media.adTrackSeconds
- media.adTrackMilestones
- media.adTrackOffsetMilestones
- media.adSegmentByMilestones
- media.adSegmentByOffsetMilestones
<table>
<thead>
<tr>
<th>Milestone</th>
<th>Milestone Syntax</th>
<th>Custom Link</th>
<th>Custom Link Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media.open</td>
<td>s.Media.open(mediaName, mediaLength, mediaPlayerName)</td>
<td>s.tl(this, linkType, linkName, variableOverrides, doneAction)</td>
<td>s.linkTrackVars = 'events, prop10, eVar10, eVar12, eVar15, contextData.video.name, contextData.video.view'; s.prop10=mediaName; s.eVar10=mediaName; s.eVar12=&quot;video&quot;; s.eVar15=mediaPlayerName; s.events='event2'; s.contextData['video.name']=mediaName; s.contextData['video.view']='true'; s.tl(this, 'o', 'Video Start');</td>
</tr>
<tr>
<td>mediaName</td>
<td>mediaName: (required) The name of the video as you want it to appear in video reports.</td>
<td>Set eVar or context data variable in link call</td>
<td>s.prop10=mediaName; s.eVar10=mediaName; s.contextData['video.name'] = mediaName;</td>
</tr>
<tr>
<td>mediaLength</td>
<td>mediaLength: (required) The length of the video in seconds.</td>
<td>Set eVar or context data variable in link call</td>
<td>s.contextData['video.length'] = ’90’;</td>
</tr>
<tr>
<td>mediaPlayerName</td>
<td>mediaPlayerName: (required) The name of the media player used to view the video, as you want it to appear in video reports.</td>
<td>Set eVar or context data variable in link call</td>
<td>s.contextData['video.player'] = ’CustomPlayer Name’;</td>
</tr>
<tr>
<td>Media.openAd</td>
<td>s.Media.openAd( name, length, playerName, parentName, parentPod, parentPodPosition, CPM)</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>name</td>
<td>name: (required) The name or ID of the ad.</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>length</td>
<td>length: (required) The length of the ad.</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>playerName</td>
<td>playerName: (required) The name of the media player used to view the ad.</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>parentName</td>
<td>parentName: The name or ID of the primary content where the ad is embedded.</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Milestone</td>
<td>Milestone Syntax</td>
<td>Custom Link</td>
<td>Custom Link Syntax</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>parentPod</td>
<td>parentPod: The position in the primary content the ad was played.</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>parentPodPosition</td>
<td>parentPodPosition: The position within the pod where the ad is played.</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>CPM</td>
<td>CPM: The CPM or encrypted CPM (prefixed with a &quot;~&quot;) that applies to this playback.</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Milestone</td>
<td>Milestone Syntax</td>
<td>Custom Link</td>
<td>Custom Link Syntax</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Media.click</td>
<td>s.Media.click(name, offset)</td>
<td>s.tl(this, linkType, linkName, variableOverrides, doneAction)</td>
<td>Use a custom link analytics call to track clicks</td>
</tr>
<tr>
<td>Media.close</td>
<td>s.Media.close(mediaName)</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.complete</td>
<td>s.Media.complete(name, offset)</td>
<td>s.tl(this, linkType, linkName, variableOverrides, doneAction)</td>
<td>s.linkTrackVars = 'events, prop10, eVar10, eVar12, eVar15, contextData.video.name, contextData.video.complete'; s.linkTrackEvents='event3'; s.prop10=mediaName; s.eVar10=mediaName; s.eVar12=&quot;video&quot;; s.eVar15=mediaPlayerName; s.events='event3'; s.contextData['video.name'] = mediaName; s.contextData['video.complete'] = 'true'; s.tl(this,'o','Video Complete');</td>
</tr>
<tr>
<td>Media.play</td>
<td>s.Media.play(name, offset, segmentNum, segment, segmentLength)</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.stop</td>
<td>s.Media.stop(mediaName, mediaOffset)</td>
<td>N/A</td>
<td>Not Available</td>
</tr>
<tr>
<td>Media.monitor</td>
<td>s.Media.monitor(s, media)</td>
<td>Set eVar or context data variable in link call</td>
<td>s.linkTrackVars = 'events, prop10, eVar10, eVar12, eVar15, contextData.video.name, contextData.video.view'; s.linkTrackEvents = 'event2';</td>
</tr>
</tbody>
</table>
Custom Link in Analytics

Custom Link implementation guide

Custom Video Tracking utilizes the manual link tracking using custom link code within Analytics appMeasurement. Most often, custom video link video tracking is used on platforms and devices where minimal video measurement is needed.

• In JavaScript: s.tl() function
• In Mobile Apps: trackAction() Android, trackAction() iOS, trackAction() OTT
• In Data Insertion API: <linktype> tag

Requirements:
• Access to video player API events and data
• Ability to add scripts if using Analytics SDK
• Ability to add tracking beacons (custom scripting or hardcode) if using Data Insertion API

Metadata:
• Metadata can be added to any tracking call as part of the link data
• Remember to update the linkTrackVars and linkTrackEvents

```javascript
/* Call on video complete */
if (e.type == "ended") {
  s.linkTrackVars = 'events, prop10, eVar10, eVar12, eVar13, eVar15';
  s.linkTrackEvents = 'event3';
  s.prop10 = mediaName;
  s.eVar10 = mediaName;
  s.eVar12 = "video";
  s.eVar13 = document.title;
  s.eVar15 = mediaPlayerName;
  s.events = 'event3';
  s.tl(this, 'o', 'Video Complete');
}
```

Why use Custom Link:
• Minimal prerequisites are needed
• Works on any platform, including no-script
• Any calculations, such as time spent or quartiles, must be calculated in a custom script
• Very straightforward with no hidden libraries or scripts
• Total control over every aspect of the video data
• Remove link to sample player

Sample JavaScript for HTML5 Player

```html
<script type="text/javascript">  myvideo = document.getElementById('movie');  myvideo.addEventListener('play', myHandler, false);  myvideo.addEventListener('seek', myHandler, false);  myvideo.addEventListener('seeking', myHandler, false);  myvideo.addEventListener('pause', myHandler, false);
</script>
```
```javascript
myvideo.addEventListener('ended', myHandler, false);

function myHandler(e) {
    var video = document.getElementsByTagName('video')[0];
    var mediaName = "13502979:Sailing";
    var mediaLength = video.duration;
    var mediaPlayerName = "HTML5 Player";

    /*Define video offset*/
    if (video.currentTime > 0) {
        mediaOffset = Math.floor(video.currentTime);
    } else {
        mediaOffset = 0;
    }

    /*Call on video start*/
    if (e.type == "play") {
        if (mediaOffset == 0) {
            console.log(mediaPlayerName + ' -> start -> playhead: ' + Math.floor(video.currentTime));
            s.linkTrackVars = 'events,prop10,eVar10,eVar12,eVar13,eVar15';
            s.linkTrackEvents = 'event2';
            s.prop10 = mediaName;
            s.eVar10 = mediaName;
            s.eVar12 = "video";
            s.eVar13 = document.title;
            s.eVar15 = mediaPlayerName;
            s.events = 'event2';
            s.tl(this, 'o', 'Video Start');
        }
    }

    /*Call on video pause*/
    if (e.type == "pause") {
        console.log(mediaPlayerName + ' -> pause -> playhead: ' + Math.floor(video.currentTime));
        if (video.currentTime != video.duration) {
            s.linkTrackVars = 'events,prop10,eVar10,eVar12,eVar13,eVar15';
            s.linkTrackEvents = 'event7';
            s.prop10 = mediaName;
            s.eVar10 = mediaName;
            s.eVar12 = "video";
            s.eVar13 = document.title;
            s.eVar15 = mediaPlayerName;
            s.events = 'event7';
            s.tl(this, 'o', 'Video Pause');
        }
    }

    /*Call on video complete*/
    if (e.type == "ended") {
        console.log(mediaPlayerName + ' -> ended -> playhead: ' + Math.floor(video.currentTime));
        s.linkTrackVars = 'events,prop10,eVar10,eVar12,eVar13,eVar15';
        s.linkTrackEvents = 'event3';
        s.prop10 = mediaName;
        s.eVar10 = mediaName;
        s.eVar12 = "video";
        s.eVar13 = document.title;
        s.eVar15 = mediaPlayerName;
        s.events = 'event3';
        s.tl(this, 'o', 'Video Complete');
    }
}
</script>
```
Introduction to Media Analytics

Prerequisites

Before you begin your tracking implementation, you have some early decisions to make, regarding which implementation makes the most sense for your situation:

• **Media Analytics** - Using the latest Media SDKs (the standard, recommended implementation) and/or the Media Collection API (RESTful)
• **Milestone** - The older Adobe tracking implementation
• **Data Insertion APIs** - Implementing tracking without using Media SDKs

For *Media Analytics*, here are the tasks you must complete prior to implementing:

1. **Enable Experience Cloud.**

   You need to implement the Experience Cloud ID Service (formerly known as Visitor ID Service).

   The Experience Cloud ID service enables the common identification framework for the Experience Cloud Core Services, solutions, and customer attributes and audiences in the People core service. It works by assigning a unique, persistent ID to a site visitor. When your organization implements the ID service, this ID lets you identify the same site visitor and their data in different Experience Cloud solutions.

   The ID service can also replace the different solution-specific IDs (for example, Analytics AID). Through the **Customer IDs and Authentication States** functionality, the ID service lets you pass in your own customer IDs to the Experience Cloud. Keep in mind, however, that the ID service only works with the solutions to which you have already subscribed. If you are not signed up for access to other products, the ID service does not provide the access.

   Going forward, the ID service is an integral component of many current and future Experience Cloud features, enhancements, and services. Currently, the ID service supports **Analytics, Audience Manager**, and **Target**.
**Important:** To participate in the Adobe Experience Cloud Device Co-op, the Experience Cloud ID service is required.

If you have not implemented the ID service, now is the time to start considering a migration strategy. For more information about the importance and role of the ID service, see *Why the Experience Cloud ID Service Should be on Your Radar.*

**Important:** In the absence of any user ID information present on the media-specific calls, the default analytics Fallback ID Methods will apply.

For additional information about the Experience Cloud ID, see *Experience Cloud ID Overview,* and *Experience Cloud ID Service.*

2. **Enable Adobe Analytics Reports.**

   To enable reports in Analytics and see the content and ad data you are collecting, see *Media reports enablement.*

## Implementation Paths

Media Analytics (Heartbeats) is Adobe’s standardized video solution. It has replaced Adobe's older Milestone model.

For each of these implementation paths, customers need to contact their Sales Representative/Account Manager to sign a new Sales Order as Media Analytics has a unique SKU and changes from a pricing model based on server calls to a model based on video streams:

- **Client Side** - These are Media Analytics-only integrations. You can choose the Media SDK and/or the Media Collection API integrations. This path can be used across any video player, including customer and/or OVP players such as Brightcove, Ooyala, thePlatform, and so on.

  If Media Analytics is your intended path, see the *Media SDK Implementation* and the *Media Collection API.*

  **Important:** To use Media Analytics, customers must also use Adobe Analytics.

- **Adobe Launch** - Adobe Launch, the follow-on product to Dynamic Tag Management, features a Media Analytics Launch Extension that facilitates implementing media tracking in your players.

  You can learn more about MA Launch here: [https://docs.adobelaunch.com/extension-reference/web/adobe-media-analytics-for-audio-and-video-extension](https://docs.adobelaunch.com/extension-reference/web/adobe-media-analytics-for-audio-and-video-extension)

- **Adobe Primetime** - Adobe Primetime is an Adobe Experience Cloud solution that helps content programmers and distributors monetize video on every connected screen.

  Primetime eliminates the complexity of reaching, monetizing, and activating global audiences across devices by providing a modular platform for video publishing, advertising, personalization, and analytics. Additionally, Primetime offers solutions and value around the following:

  - Support for accurately measuring Linear and VOD content types.
  - Support for measuring ad breaks with (or without) dynamic ad insertion.
  - TVSDK’s seamless ad insertion model allows for analytics that directly measures the ad playback, which increases accuracy.
• Robust set of events and metadata to ensure accuracy across QoS buffering or mobile connectivity interruptions issues and end-user interactions such as seeking, pausing, and backgrounding on mobile.
• Integrated support for Nielsen DTVR (linear) with ID3 metadata and DCR with CMS metadata.

TVSDK is already integrated with the Video Analytics (Heartbeats) SDK, which makes implementation much easier and faster across every supported platform. Primetime also supports the partnership with Nielsen. To leverage Primetime, follow the same guidelines and prerequisites found in Client-side along with the following docs for your platform(s):

• Media Analytics in TVSDK 1.4 for Android
• Media Analytics in TVSDK 2.4 for Browser TVSDK
• Media Analytics in TVSDK 1.4 for Desktop HLS
• Media Analytics in TVSDK 2.3 for Desktop HLS
• Media Analytics in TVSDK 1.4 for iOS

You should also contact your Sales Representative/Account Manager to discuss what you need to do to purchase TVSDK.

Client-side

Media Analytics (Heartbeats) is Adobe's standardized media tracking solution. It has replaced Adobe's older Milestone model.

Client Side integrations are implemented with Media Analytics only. You can choose to implement either the Media SDK or the Media Collection API (or both). The client-side path can be used across any player, including customer and/or OVP players such as Brightcove, Ooyala, thePlatform, and so on.

If Media Analytics is your intended path, see Media SDK Implementation and Media Collection API.

⚠️ Important: To use Media Analytics, customers must also use Adobe Analytics.

💡 Note:

Customers need to contact their Sales Representative/Account Manager to sign a new Sales Order as Media Analytics has a unique SKU and changes from a pricing model based on server calls to a model based on media streams.

Adobe Launch

Media Analytics (Heartbeats) is Adobe's standardized media tracking solution. It has replaced Adobe's older Milestone model.

Adobe Launch, the follow-on product to Dynamic Tag Management, features a Launch Extension that facilitates implementing tracking in your media players.

You can learn more about the Launch extension here: Adobe Analytics for Media Extension

💡 Note: For this implementation path, customers need to contact their Sales Representative/Account Manager to sign a new Sales Order as Media Analytics has a unique SKU and changes from a pricing model based on server calls to a model based on media streams.
**Primetime**

Media Analytics (Heartbeats) is Adobe's standardized media tracking solution. It has replaced Adobe's older Milestone model.

Adobe Primetime is an Adobe Experience Cloud solution that helps content programmers and distributors monetize media on every connected screen.

Primetime eliminates the complexity of reaching, monetizing, and activating global audiences across devices by providing a modular platform for media publishing, advertising, personalization, and analytics. Additionally, Primetime offers solutions and value around the following:

- Support for accurately measuring Linear and VOD content types.
- Support for measuring ad breaks with (or without) dynamic ad insertion.
- TVSDK's seamless ad insertion model allows for analytics that directly measures the ad playback, which increases accuracy.
- Robust set of events and metadata to ensure accuracy across QoS buffering or mobile connectivity interruptions issues and end-user interactions such as seeking, pausing, and backgrounding on mobile.
- Integrated support for Nielsen DTVR (linear) with ID3 metadata and DCR with CMS metadata.

TVSDK is already integrated with the Media SDK, which makes implementation much easier and faster across every supported platform. Primetime also supports the partnership with Nielsen. To leverage Primetime, follow the same guidelines and prerequisites found in Client-side along with the following docs for your platform(s):

- Media Analytics in TVSDK 1.4 for Android
- Media Analytics in TVSDK 2.4 for Browser TVSDK
- Media Analytics in TVSDK 1.4 for Desktop HLS
- Media Analytics in TVSDK 2.3 for Desktop HLS
- Media Analytics in TVSDK 1.4 for iOS

**Note:** For this implementation path, customers need to contact their Sales Representative/Account Manager to sign a new Sales Order, as Media Analytics has a unique SKU and changes from a pricing model based on server calls to a model based on media streams. You should also contact your Sales Representative/Account Manager to discuss purchasing TVSDK.

**Audience Manager enablement**

Adobe Audience Manager (AAM), a Data Management Platform (DMP), helps you bring your audience data assets together, making it easy to collect commercially relevant information about site visitors, create marketable segments, and serve targeted advertising and content to the right audience.

With AAM, you are not tied to a data seller, exchange, or demand-side platform. Additionally, AAM is completely agnostic when it comes to your partners’ data assets. With access to multiple data sources, AAM offers digital publishers the ability to use a wide variety of third-party data and our private data co-op. To learn more about AAM, see the AAM documentation **Overview**.

**VA to AAM data transfer** - For both video content and video ads, the metrics and metadata that are collected by using solution (reserved) variables can be automatically sent to AAM. The data transfer is available across all
platforms including desktop, mobile, and OTT. To enable this server-side data transfer, you need to reach out to Adobe Client Care and ask for this feed to be enabled.

**Important:** To ensure the smooth transfer of data to AAM, you should be on the latest versions of the heartbeat libraries.

Federated Data fully supports data sharing to AAM. Please work with your Adobe team for confirmation of Federated Data settings.

**OTT / AAM methods**

You can use these methods to send signals and retrieve visitor segments from Audience Manager:

- **Chromecast:**
  - `getVisitorProfile()` -
    Returns the visitor profile that was most recently obtained. Returns an empty object if no signal has been submitted yet.
    ```javascript
    ADBMobile.audienceManager.getVisitorProfile();
    ```
  - `getDpid()` -
    Returns the visitor profile that was most recently obtained. Returns an empty object if no signal has been submitted yet.
    ```javascript
    ADBMobile.audienceManager.getDpid();
    ```
  - `getDpuuid()` -
    Returns the current DPUUID.
    ```javascript
    ADBMobile.audienceManager.getDpuuid();
    ```
  - `setDpidAndDpuuid()` -
    Sets the DPID and DPUUID. If DPID and DPUUID are set, they will be sent with each signal.
    ```javascript
    ADBMobile.audienceManager.setDpidAndDpuuid("myDpid", "myDpuuid");
    ```
  - `submitSignal()` -
    Sends audience management a signal with traits.
    ```javascript
    ADBMobile.audienceManager.submitSignal();
    ```

- **Roku:**
  - `audienceVisitorProfile` -
    Returns the visitor profile that was most recently obtained. Returns an empty object if no signal has been submitted yet.
    ```javascript
    ADBMobile().audienceVisitorProfile()
    ```
  - `audienceDpid` -
    Returns the visitor profile that was most recently obtained. Returns an empty object if no signal has been submitted yet.
    ```javascript
    ADBMobile().audienceDpid()
    ```
• audienceDpuuid -
  Returns the current DPUUID.
  ADBMobile().audienceDpuuid()

• audienceSetDpidAndDpuuid -
  Sets the DPID and DPUUID. If DPID and DPUUID are set, they will be sent with each signal.
  ADBMobile().audienceSetDpidAndDpuuid("myDpid", "myDpuuid")

• audienceSubmitSignal -
  Sends audience management a signal with traits.
  ADBMobile().audienceSubmitSignal()
Media SDK

Download SDKs

Download the 2.x Media SDKs

<table>
<thead>
<tr>
<th>2.x Media SDKs</th>
<th>Latest Media SDK Downloads</th>
<th>APIs</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android/FireTV</td>
<td>Media SDK for Android v2.2.0</td>
<td>Android API Reference</td>
<td>Set up Android</td>
</tr>
<tr>
<td>iOS/AppleTV</td>
<td>Media SDK for iOS v2.2.0</td>
<td>iOS API Reference</td>
<td>Set up iOS</td>
</tr>
<tr>
<td>(Contact Customer Care to obtain legacy 2.0 SDK)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaScript</td>
<td>Media SDK for JS v2.2.0</td>
<td>JS API Reference</td>
<td>Set up JavaScript</td>
</tr>
<tr>
<td>Roku</td>
<td>Media SDK for Roku v2.2.1</td>
<td></td>
<td>Set Up Roku</td>
</tr>
<tr>
<td>Chromecast</td>
<td>Media SDK for Chromecast v2.2.0</td>
<td>Chromecast API Reference</td>
<td>Set up Chromecast</td>
</tr>
</tbody>
</table>

Download the Adobe Nielsen 2.x SDKs

<table>
<thead>
<tr>
<th>Adobe Nielsen 2.x SDKs</th>
<th>Latest Media SDK Downloads</th>
<th>Nielsen Implementation Guides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>VHL for Android v.2.0.1N</td>
<td>2.1 for Android</td>
</tr>
<tr>
<td>iOS</td>
<td>VHL for iOS v.2.0.1N</td>
<td>2.1 for iOS</td>
</tr>
<tr>
<td>JavaScript</td>
<td>VHL for JavaScript v.2.0.1N</td>
<td>2.1 for JavaScript</td>
</tr>
</tbody>
</table>

Media Analytics Launch Extension

<table>
<thead>
<tr>
<th>MA Launch Component</th>
<th>Downloads</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA Launch Extension</td>
<td>Sample HTML5 Player Extension</td>
<td>Launch Extension Reference</td>
</tr>
</tbody>
</table>

Download the 1.x Media SDKs

<table>
<thead>
<tr>
<th>1.x SDKs</th>
<th>Media SDK Downloads</th>
<th>Getting Started Guides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>SDK for Android v1.5.8</td>
<td>Get Started - Media 1.x for Android</td>
</tr>
<tr>
<td>iOS</td>
<td>SDK for iOS v1.5.9</td>
<td>Get Started - Media 1.x for iOS</td>
</tr>
<tr>
<td>JavaScript</td>
<td>SDK for JavaScript v1.5.7</td>
<td>Get Started - Media 1.x for JS</td>
</tr>
<tr>
<td>TVML</td>
<td>SDK for TVML v1.0.0</td>
<td>Get Started - Media 1.x for TVML</td>
</tr>
<tr>
<td>1.x SDKs</td>
<td>Media SDK Downloads</td>
<td>Getting Started Guides</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><strong>Apple TV</strong></td>
<td>SDK for Apple TV v1.0.0</td>
<td>Get Started - Media 1.x for Apple TV</td>
</tr>
<tr>
<td>(Contact Customer Care to obtain legacy 1.x SDK)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Chromecast</strong></td>
<td>SDK for Chromecast v1.0.0</td>
<td>Get Started - Media 1.x for Chromecast</td>
</tr>
<tr>
<td>(Contact Customer Care to obtain legacy 1.x SDK)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Set up and configure

### Overview

⚠️ **Important:** The following instructions are for the 2.x Media SDKs. If you are implementing a 1.x version of the Media SDK, see the 1.x Media SDK Documentation. For Primetime integrators, see Primetime Media SDK Documentation.

### Minimum Platform Version Support

The following table describes the minimum platform versions supported for each SDK, beginning February 19, 2019.

<table>
<thead>
<tr>
<th>OS/Browser</th>
<th>Min Version Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS</td>
<td>iOS 6+</td>
</tr>
<tr>
<td>Android</td>
<td>Android 5.0+ - Lollipop</td>
</tr>
<tr>
<td>Chrome</td>
<td>v22+</td>
</tr>
<tr>
<td>Mozilla</td>
<td>v27+</td>
</tr>
<tr>
<td>Safari</td>
<td>v7+</td>
</tr>
<tr>
<td>IE</td>
<td>v11+</td>
</tr>
</tbody>
</table>

### General Implementation Guidelines

There are three main SDK components involved in media tracking:

- Media Heartbeat Config - The config contains the basic settings for reporting.
- Media Heartbeat Delegate - The delegate controls playback time and the QoS object.
- Media Heartbeat - The primary library containing members and methods.

1. Create a MediaHeartbeatConfig instance and set your config values.
Table 2: Config parameters:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>trackingServer</td>
<td>Tracking server for media analytics. This is different from your analytics tracking server.</td>
<td>Yes</td>
<td>Empty String</td>
</tr>
<tr>
<td>channel</td>
<td>Channel name</td>
<td>No</td>
<td>Empty String</td>
</tr>
<tr>
<td>ovp</td>
<td>Name of the online media platform through which content gets distributed</td>
<td>No</td>
<td>Empty String</td>
</tr>
<tr>
<td>appVersion</td>
<td>Version of the media player app/SDK</td>
<td>No</td>
<td>Empty String</td>
</tr>
<tr>
<td>playerName</td>
<td>Name of the media player in use, i.e., “AVPlayer”, “HTML5 Player”, “My Custom Player”</td>
<td>No</td>
<td>Empty String</td>
</tr>
<tr>
<td>ssl</td>
<td>Indicates whether calls should be made over HTTPS</td>
<td>No</td>
<td>false</td>
</tr>
<tr>
<td>debugLogging</td>
<td>Indicates whether debug logging is enabled</td>
<td>No</td>
<td>false</td>
</tr>
</tbody>
</table>

2. Implement the `MediaHeartbeatDelegate`.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>getQoSObject()</td>
<td>Returns the <code>MediaObject</code> instance that contains the current QoS information. This method will be called multiple times during a playback session. Player implementation must always return the most recently available QoS data.</td>
<td>Yes</td>
</tr>
<tr>
<td>getCurrentPlaybackTime()</td>
<td>Returns the current position of the playhead.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>For VOD tracking, the value is specified in seconds from the beginning of the media item.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For LINEAR/LIVE tracking, the value is specified in seconds from the beginning of the program.</td>
<td></td>
</tr>
</tbody>
</table>

**Tip:** The Quality of Service (QoS) object is optional. If QoS data is available for your player and you wish to track that data, then the following variables are required:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate</td>
<td>The bitrate of media in bits per second.</td>
<td>Yes</td>
</tr>
<tr>
<td>startupTime</td>
<td>The start up time of media in milliseconds.</td>
<td>Yes</td>
</tr>
<tr>
<td>fps</td>
<td>The frames displayed per second.</td>
<td>Yes</td>
</tr>
<tr>
<td>droppedFrames</td>
<td>The number of dropped frames so far.</td>
<td>Yes</td>
</tr>
</tbody>
</table>
3. Create the MediaHeartbeat instance.

Use the MediaHeartbeatConfig and MediaHeartbeatDelegate to create the MediaHeartbeat instance.

⚠️ **Important:** Make sure that your MediaHeartbeat instance is accessible and does not get deallocated until the end of the session. This instance will be used for all the following media tracking events.

💡 **Tip:** MediaHeartbeat requires an instance of AppMeasurement to send calls to Adobe Analytics.

4. Combine all of the pieces.

The following sample code utilizes our JavaScript 2.x SDK for an HTML5 video player:

```javascript
// Create local references to the heartbeat classes
var MediaHeartbeat = ADB.va.MediaHeartbeat;
var MediaHeartbeatConfig = ADB.va.MediaHeartbeatConfig;
var MediaHeartbeatDelegate = ADB.va.MediaHeartbeatDelegate;

// Media Heartbeat Config
var mediaConfig = new MediaHeartbeatConfig();
mediaConfig.trackingServer = "namespace.hb.omtrdc.net";
mediaConfig.playerName = "HTML5 Basic";
mediaConfig.channel = "Video Channel";
mediaConfig.debugLogging = true;
mediaConfig.appVersion = "2.0";
mediaConfig.ssl = false;
mediaConfig.ovp = "";

// Media Heartbeat Delegate
var mediaDelegate = new MediaHeartbeatDelegate();

// Set mediaDelegate CurrentPlaybackTime
mediaDelegate.getCurrentPlaybackTime = function() {
    return video.currentTime;
};

// Set mediaDelegate QoSObject - OPTIONAL
mediaDelegate.getQoSObject = function() {
    return MediaHeartbeat.createQoSObject(video.bitrate,
                                            video.startuptime,
                                            video.fps,
                                            video.droppedframes);
}

// Create mediaHeartbeat instance
this.mediaHeartbeat = new MediaHeartbeat(mediaDelegate, mediaConfig, appMeasurementInstance);
```

**Validate**

Media implementations are composed of two types of tracking calls:

- Media and Ad Start calls are sent directly to the AppMeasurement server.
- Heartbeat calls are sent to the Heartbeat tracking server on start, every ten seconds for content, and every one second for ads.

Media tracking works the same across all platforms, desktop and mobile. Audio tracking currently works in mobile platforms. For all tracking calls there are a few key universal variables to be validated:

- **AppMeasurement (Analytics)**
For more information about tracking server options, see Correctly populate the trackingServer and trackingServerSecure variable.

⚠️ **Important:** An RDC tracking server or CNAME resolving to an RDC server is required for Experience Cloud Visitor ID service.

The analytics tracking server should end in `.sc.omtrdc.net` or be a CNAME.

**• Heartbeats (Media Analytics)**

Always has the format `[namespace].hb.omtrdc.net`, where `[namespace]` is defined by your login company and is provided by Adobe.

### 1.x SDK Documentation

<table>
<thead>
<tr>
<th>Media Analytics 1.x SDKs*</th>
<th>Developer Guides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>Configure for Android</td>
</tr>
<tr>
<td>AppleTV</td>
<td>Configure for AppleTV</td>
</tr>
<tr>
<td>Chromecast</td>
<td>Configure for Chromecast</td>
</tr>
<tr>
<td>iOS</td>
<td>Configure for iOS</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Configure for JavaScript</td>
</tr>
</tbody>
</table>
| **Primetime**            | • **Android:** Configure Video Analytics  
                          | • **DHLS:** Configure Video Analytics  
                          | • **iOS:** Configure Video Analytics |
| TVML                     | Configure for TVML |

* For all 1.x SDKs, the links are for the full PDF download of the documentation.

### Primetime Media SDK Documentation

**• Android:** Configure Media Analytics  
**• DHLS:** Configure Media Analytics  
**• iOS:** Configure Media Analytics

### Set up Android

**• Obtain valid configuration parameters for the Media SDK** - These parameters can be obtained from an Adobe representative after you set up your analytics account.

**• Implement ADBMobile for Android in your application** - For more information about the Adobe Mobile SDK documentation, see [Android SDK 4.x for Experience Cloud Solutions](#).

**• Provide the following capabilities in your media player:**

  - An API to subscribe to player events - The Media SDK requires that you call a set of simple APIs when events occur in your player.
  - An API that provides player information - This information includes details such as the media name and the play head position.

1. Add your downloaded Media SDK to your project.
1. Expand the Android zip file (e.g., MediaSDK-android-v2.*.zip).
2. Verify that the MediaSDK.jar file exists in the libs/ directory.
3. Add the library to your project.

**IntelliJ IDEA:**

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 MediaSDK.jar file.
6. Select the modules in which you plan to use the mobile library.
7. Click **Apply** and then **OK** to close the Module Settings window.

**Eclipse:**

1. In the Eclipse IDE, right-click on the project name.
2. Click **Build Path > Add External Archives**.
3. Select **MediaSDK.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 MediaSDK.jar file is selected.

2. Import the library.

   ```java
   import com.adobe.primetime.va.simple.MediaHeartbeat;
   import com.adobe.primetime.va.simple.MediaHeartbeatDelegate;
   import com.adobe.primetime.va.simple.MediaHeartbeatConfig;
   import com.adobe.primetime.va.simple.MediaObject;
   ```

3. Create the MediaHeartbeatConfig instance.

   **Here is a sample MediaHeartbeatConfig initialization:**

   ```java
   // Media Heartbeat Initialization
   config.trackingServer = <SAMPLE_HEARTBEAT_TRACKING_SERVER>;
   config.channel = <SAMPLE_HEARTBEAT_CHANNEL>;
   config.appVersion = <SAMPLE_HEARTBEAT_SDK_VERSION>;
   config.ovp = <SAMPLE_HEARTBEAT_OVP_NAME>;
   config.playerName = <SAMPLE_PLAYER_NAME>;
   config.ssl = <true/false>;
   config.debugLogging = <true/false>;
   ```

4. Implement the MediaHeartbeatDelegate interface.

   ```java
   public class VideoAnalyticsProvider
       implements Observer, MediaHeartbeatDelegate {}
   ```

   // Replace <bitrate>, <startupTime>, <fps>, and // <droppeFrames> with the current playback QoS values.
   @Override
   public MediaObject getQoSObject() {
       return MediaHeartbeat.createQoSObject(<bitrate>,
                                              <startupTime>,
                                              <fps>,
                                              <droppedFrames>);
   }

   //Replace <currentPlaybackTime> with the video player current playback time
5. Create the MediaHeartbeat instance.

Use the MediaHeartbeatConfig instance and the MediaHeartbeatDelegate instance to create the MediaHeartbeat instance.

```java
// Replace <MediaHeartbeatDelegate> with your delegate instance
MediaHeartbeat _heartbeat =
    new MediaHeartbeat(<MediaHeartbeatDelegate>, config);
```

**Important:** Make sure that your MediaHeartbeat instance is accessible and does not get deallocated until the end of the session. This instance will be used for all of the following tracking events.

### Adding app permissions

Your app using the Media SDK 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:

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

### Migrating from version 1.x to 2.x in Android

In versions 2.x, all of the public methods are consolidated into the com.adobe.primetime.va.simple.MediaHeartbeat class to make it easier on developers. Also, all configs are now consolidated into the com.adobe.primetime.va.simple.MediaHeartbeatConfig class.

For detailed information about migrating from 1.x to 2.x, see Media Analytics 1.x to 2.x migration.

### Set up iOS

- **Obtain valid configuration parameters for the Media SDK** - These parameters can be obtained from an Adobe representative after you set up your analytics account.

- **Implement ADBMobile for iOS in your application** - For more information about the Adobe Mobile SDK documentation, see iOS SDK 4.x for Experience Cloud Solutions.

**Important:** Beginning with iOS 9, Apple introduced a feature called App Transport Security (ATS). This feature aims to improve network security by ensuring that your apps use only industry-standard protocols and ciphers. This feature is enabled by default, but you have configuration options that provide you with choices for working with ATS. For details on ATS, see https://marketing.adobe.com/resources/help/en_US/mobile/ios/app_transport_security.html.

- **Provide the following capabilities in your media player:**
  
  - **An API to subscribe to player events** - The Media SDK requires that you call a set of simple APIs when events occur in your player.
• An API that provides player information - This information includes details such as the media name and the play head position.

1. Add your downloaded Media SDK to your project.

1. Verify that the following software components exist in the libs directory:
   • ADBMediaHeartbeat.h: The Objective-C header file that is used for iOS heartbeat tracking APIs.
   • ADBMediaHeartbeatConfig.h: The Objective-C header file for the SDK configuration.
   • MediaSDK.a: A bitcode-enabled fat binary that contains the library builds for iOS devices (armv7, armv7s, arm64) and simulators (i386 and x86_64).
     This binary should be linked when the target is intended for an iOS app.
   • MediaSDK_TV.a: A bitcode-enabled fat binary containing the library builds for new Apple TV devices (arm64) and simulator (x86_64).
     This binary should be linked when the target is intended for an Apple TV (tvOS) app.

2. Add the library to your project:

   a. Launch the Xcode IDE and open your app.
   b. In Project Navigator, drag the libs directory and drop it under your project.
   c. Ensure that the Copy Items if Needed checkbox is selected, the Create Groups is selected, and none of the checkboxes in Add to Target are selected.
   d. Click Finish.
   e. In Project Navigator, select your app and select your targets.
   f. Link the required frameworks and libraries in the Linked Frameworks and Libraries section on the General tab.

   iOS App Targets:
   • AdobeMobileLibrary.a
• MediaSDK.a
• libsqlite3.0.tbd

Apple TV (tvOS) Targets:
• AdobeMobileLibrary_TV.a
• MediaSDK_TV.a
• libsqlite3.0.tbd
• SystemConfiguration.framework

g. Verify that your app builds without errors.

2. Import the library.

``` objective-c
#import "ADBMediaHeartbeat.h"
#import "ADBMediaHeartbeatConfig.h"
```

3. Create a ADBMediaHeartbeatConfig instance.

This section helps you to understand the MediaHeartbeat config parameters, and to set correct config values on your MediaHeartbeat instance for accurate tracking.

Here is a sample ADBMediaHeartbeatConfig initialization:

``` objective-c
// Media Heartbeat Initialization
ADBMediaHeartbeatConfig *config = [[ADBMediaHeartbeatConfig alloc] init];
config.trackingServer = <SAMPLE_HEARTBEAT_TRACKING_SERVER>;
config.channel = <SAMPLE_HEARTBEAT_CHANNEL>;
config.appVersion = <SAMPLE_HEARTBEAT_SDK_VERSION>;
config.ovp = <SAMPLE_HEARTBEAT_OVP_NAME>;
config.playerName = <SAMPLE_PLAYER_NAME>;
config.ssl = <YES/NO>;
config.debugLogging = <YES/NO>;
```

4. Implement the ADBMediaHeartbeatDelegate protocol.

``` objective-c
@interface VideoAnalyticsProvider : NSObject <ADBMediaHeartbeatDelegate>
@end
@implementation VideoAnalyticsProvider

// Replace <bitrate>, <startupTime>, <fps> and <droppeFrames>
// with the current playback QoS values.
- (ADBMediaObject *)getQoSObject {
    return [ADBMediaHeartbeat createQoSObjectWithBitrate:<bitrate>
        startupTime:<startupTime>
        fps:<fps>
        droppedFrames:<droppedFrames>];
}

// Return the current video player playhead position.
// Replace <currentPlaybackTime> with the video player current playback time
- (NSTimeInterval)getCurrentPlaybackTime {
    return <currentPlaybackTime>;
}
@end
```

5. Use the ADBMediaHeartBeatConfig and ADBMediaHeartBeatDelegate to create the ADBMediaHeartbeat instance.

``` objective-c
//Replace <ADBMediaHeartBeatDelegate> with your delegate instance
_mediarHeartbeat = [[ADBMediaHeartbeat alloc] initWithDelegate:
    <ADBMediaHeartBeatDelegate> config:config];
```
**Important:** Make sure that your ADBMediaHeartbeat instance is accessible and does not get deallocated until the end of the session. This instance will be used for all the following tracking events.

**Migrating from version 1.x to 2.x in iOS**

In version 2.x, all of the public methods are consolidated into the ADBMediaHeartbeat class to make it easier on developers. All configurations have been consolidated into the ADBMediaHeartbeatConfig class.

For more information about migrating from 1.x to 2.x, see Media Analytics 1.x to 2.x Migration.

**Configure a Native App for tvOS -**

With the release of the new Apple TV, you can now create applications to run in the native tvOS environment. You can either create a purely native app, using any of several frameworks available in iOS, or you can create your app using XML templates and JavaScript. Starting with MediaSDK version 2.0, support for tvOS is available. For more information about tvOS, see tvOS Developer site.

Perform the following steps in your Xcode project. This guide is written assuming your project has a target that is an Apple TV app targeting tvOS:

1. Drag the VideoHeartbeat_TV.a library file into your project’s lib folder.
2. In the Build Phases tab of your tvOS app’s target, expand the Link Binary with Libraries section and add the following libraries:
   - MediaSDK_TV.a
   - AdobeMobileLibrary_TV.a
   - libsqlite3.0.tbd
   - SystemConfiguration.framework

**Set up JavaScript**

- **Obtain valid configuration parameters** - These parameters can be obtained from an Adobe representative after you set up your analytics account.
- **Implement AppMeasurement for JavaScript in your media application** - See Implementing Analytics Using JavaScript.
- **Provide the following capabilities in your media player:**
  - **An API to subscribe to player events** - The Media SDK requires that you call a set of simple APIs when events occur in your player.
  - **An API that provides player information** - This information includes details such as the media name and the play head position.

1. Add your downloaded library to your project. Create local references to the classes for convenience.
   1. Expand the MediaSDK-js-v2.*.zip file that you downloaded.
   2. Verify that the MediaSDK.min.js file exists in the libs directory:
   3. Host the MediaSDK.min.js file.
      - This core JavaScript file must be hosted on a web server that is accessible to all pages on your site. You need the path to these files for the next step.
   4. Reference MediaSDK.min.js on all site pages.
Include MediaSDK for JavaScript by adding the following line of code in the `<head>` or `<body>` tag on each page. For example:

```html
<script type="text/javascript"
src="http://INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/MediaSDK.min.js"></script>
```

5. To quickly verify that the library was successfully imported, instantiate the `ADB.va.MediaHeartbeatConfig` class.

   **Note:** From Version 2.1.0, the JavaScript SDK is compliant with the AMD and CommonJS module specifications, and `MediaSDK.min.js` can also be used with compatible module loaders.

2. For easy access to the APIs, create local references to the `MediaHeartbeat` classes.

   var MediaHeartbeat = ADB.va.MediaHeartbeat;
   var MediaHeartbeatConfig = ADB.va.MediaHeartbeatConfig;
   var MediaHeartbeatDelegate = ADB.va.MediaHeartbeatDelegate;

3. Create a `MediaHeartbeatConfig` instance.

   This section helps you to understand `MediaHeartbeat` config parameters and how to set correct config values on your `MediaHeartbeat` instance, for accurate tracking.

   **Here is a sample `MediaHeartbeatConfig` initialization:**

   ```javascript
   //Media Heartbeat initialization
   var mediaConfig = new MediaHeartbeatConfig();
   mediaConfig.trackingServer = Configuration.HEARTBEAT.TRACKING_SERVER;
   mediaConfig.playerName = Configuration.PLAYER.NAME;
   mediaConfig.channel = Configuration.HEARTBEAT.CHANNEL;
   mediaConfig.debugLogging = true;
   mediaConfig.appVersion = Configuration.HEARTBEAT.SDK;
   mediaConfig.ssl = false;
   mediaConfig.ovp = Configuration.HEARTBEAT.OVP;
   ```

4. Implement the `MediaHeartbeatDelegate` protocol.

   var mediaDelegate = new MediaHeartbeatDelegate();

   ```javascript
   // Replace `<currentPlaybackTime>` with the video player current playback time
   mediaDelegate.getCurrentPlaybackTime = function() {
     return <currentPlaybackTime>;
   };

   // Replace `<bitrate>`, `<startTime>`, `<fps>` and `<droppedFrames>` with the current playback QoS values.
   mediaDelegate.getQoSObject = function() {
     return MediaHeartbeat.createQoSObject(<bitrate>, <startTime>, <fps>, <droppedFrames>);
   };
   ```

5. Create the `MediaHeartbeat` instance.

   Use the `MediaHeartbeatConfig` and `MediaHeartbeatDelegate` to create the `MediaHeartbeat` instance.

   ```javascript
   this.mediaHeartbeat = new MediaHeartbeat(mediaDelegate, mediaConfig, appMeasurement);
   ```

   **Important:** Make sure that your `MediaHeartbeat` instance is accessible and does not get deallocated until the end of the media session. This instance will be used for all of the following tracking events.

   **Tip:** `MediaHeartbeat` requires an instance of `AppMeasurement` to send calls to Adobe Analytics.

   **Here is an example of an `AppMeasurement` instance:**

   ```javascript
   // AppMeasurement instance example
   var appMeasurement = new AppMeasurement();
   ```
Migrate from version 1.x to 2.x in JavaScript

In version 2.x, all of the public methods are consolidated into the `ADB.va.MediaHeartbeat` class to make it easier on developers. Also, all configs are now consolidated into the `ADB.va.MediaHeartbeatConfig` class.

For detailed information about migrating from 1.x to 2.x, see Media Analytics 1.x to 2.x Migration.

Set up Chromecast

Should I use the Chromecast JavaScript SDK or can I use the Standard JavaScript SDK? The correct answer is “Chromecast”, for these reasons:

• The AppMeasurement and VisitorAPI libraries in the Standard JS SDK are not certified to work on OTT platforms. In the Chromecast JS SDK, the Video Heartbeats Library (VHL), Analytics, and VisitorAPI are all built-in to the single, unified, certified-for-Chromecast SDK.

• The Chromecast SDK is much more lightweight than the standard JS SDK. This is very crucial for the lower-end hardware used by OTT platforms.

• Obtain valid configuration parameters for Heartbeats - These parameters can be obtained from an Adobe representative after you set up your media analytics account.

• Provide the following capabilities in your media player:

  • An API to subscribe to player events - The Media SDK requires that you call a set of simple APIs when events occur in your player.

  • An API that provides player information - This information includes details such as the media name and the play head position.

Adobe Mobile services provides a new UI that brings together mobile marketing capabilities for mobile applications from across the Adobe Marketing Cloud. Initially, the Mobile service provides seamless integration of app analytics and targeting capabilities for the Adobe Analytics and Adobe Target solutions. Learn more at Adobe Mobile Services documentation.

Chromecast SDK 2.x for Experience Cloud Solutions lets you measure Chromecast applications written in JavaScript, leverage and collect audience data through audience management, and measure video engagement through Video heartbeats.

1. Add your downloaded Chromecast library to your project.

   1. The `AdobeMobileLibrary-Chromecast-2.1.0` zip file consists of the following software components:

      • `adbmobile-chromecast.min.js`:

      This library file will be included in your Chromecast app source folder.

      • `ADBMobileConfig config`

      This SDK configuration file is customized for your app. A sample `ADBMobileConfig` implementation is provided with the SDK (under `samples/`). Obtain the proper settings from an Adobe representative.
2. Add the library file to your index.html file, and create the ADBMobileConfig global variable as follows (the global variable used to configure Adobe Mobile for Heartbeats has an exclusive key named mediaHeartbeat):

```javascript
<script>
    var ADBMobileConfig = {
        "marketingCloud": {
            "org": "972C898555E9F7BC7F000101@AdobeOrg"
        },
        "target": {
            "clientCode": ",",
            "timeout": 5
        },
        "audienceManager": {
            "server": "obumobile5.demdex.net"
        },
        "analytics": {
            "rsids": "mobile5vh1.sample.player",
            "server": "obumobile5.sc.omtrdc.net",
            "ssl": false,
            "offlineEnabled": false,
            "charset": "UTF-8",
            "lifecycleTimeout": 300,
            "privacyDefault": "optedin",
            "batchLimit": 0,
            "timezone": "MDT",
            "timezoneOffset": -360,
            "referrerTimeout": 0,
            "poi": []
        },
        "mediaHeartbeat": {
            "server": "obumobile5.hb.omtrdc.net",
            "publisher": "972C898555E9F7BC7F000101@AdobeOrg",
            "channel": "test-channel-chromecast",
            "ssl": false,
            "ovp": "chromecast-player",
            "sdkVersion": "chromecast-sdk",
            "playerName": "Chromecast"
        }
    };
</script>

Table 3: ADBMobile Config Parameters for mediaHeartbeat key:

<table>
<thead>
<tr>
<th>Config Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>String that represents the URL of the tracking endpoint on the backend.</td>
</tr>
<tr>
<td>publisher</td>
<td>String that represents the content publisher unique identifier.</td>
</tr>
<tr>
<td>channel</td>
<td>String that represents the name of the content distribution channel.</td>
</tr>
<tr>
<td>ssl</td>
<td>Boolean that represents whether SSL should be used for tracking calls.</td>
</tr>
<tr>
<td>ovp</td>
<td>String that represents the name of the video player provider.</td>
</tr>
<tr>
<td>sdkversion</td>
<td>String that represents the current version of the app/SDK.</td>
</tr>
<tr>
<td>Config Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>playerName</td>
<td>String that represents the name of the player.</td>
</tr>
</tbody>
</table>

**Important:** If mediaHeartbeat is incorrectly configured, the media module (VHL) enters an error state and will stop sending tracking calls.

2. **Configure Experience Cloud Visitor ID.**

The Experience Cloud Visitor ID service provides a universal Visitor ID across Experience Cloud solutions. The Visitor ID service is required by Video heartbeat and other Marketing Cloud integrations.

Verify that your `ADBMobileConfig` config contains your `marketingCloud` organization ID.

```json
"marketingCloud": {
  "org": YOUR-MCORG-ID"
}
```

Experience Cloud organization IDs uniquely identify each client company in the Adobe Marketing Cloud and appear similar to the following value: 016D5C175213CCA80A490D05@AdobeOrg.

**Important:** Ensure that you include @AdobeOrg.

After the configuration is complete, an Experience Cloud Visitor ID is generated and is included on all hits. Other Visitor IDs, such as custom and automatically-generated, continue to be sent with each hit.

**Experience Cloud Visitor ID Service Methods**

**Tip:** Experience Cloud Visitor ID methods are prefixed with `visitor`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getMarketingCloudID()</td>
<td>Retrieves the Experience Cloud Visitor ID from the Visitor ID service. <code>ADBMobile.visitor.getMarketingCloudID()</code></td>
</tr>
</tbody>
</table>
| syncIdentifiers()           | With the Experience Cloud Visitor ID, you can set additional customer IDs that can be associated with each visitor. The Visitor API accepts multiple customer IDs for the same visitor and a customer type identifier to separate the scope of the different customer IDs. This method corresponds to `setCustomerIDs()` in the JavaScript library. For example: ```javascript
var identifiers = {};
identifiers["idType"] = "idValue";
ADBMobile.visitor.syncIdentifiers(identifiers);``` |

**Postbacks** - For more information about configuring postbacks, see Configure Postbacks.
Set Up Roku

• **Obtain valid configuration parameters for Heartbeats** - These parameters can be obtained from an Adobe representative after you set up your media analytics account.

• **Provide the following capabilities in your media player:**
  
  • *An API to subscribe to player events* - The Media SDK requires that you call a set of simple APIs when events occur in your player.
  
  • *An API that provides player information* - This information includes details such as the media name and the play head position.

Adobe Mobile services provides a new UI that brings together mobile marketing capabilities for mobile applications from across the Adobe Marketing Cloud. Initially, the Mobile service provides seamless integration of app analytics and targeting capabilities for the Adobe Analytics and Adobe Target solutions.

Learn more at [Adobe Mobile Services documentation](#).

Roku SDK 2.x for Experience Cloud Solutions lets you measure Roku applications written in BrightScript, leverage and collect audience data through audience management, and measure video engagement through Video heartbeats.

1. Add your **downloaded** Roku library to your project.

   1. The AdobeMobileLibrary-2.*-Roku.zip download file consists of the following software components:
      
      • `adbmobile.brs`:
        
        This library file will be included in your Roku app source folder.

      • `ADBMobileConfig.json`
        
        This SDK configuration file is customized for your app.

   2. Add the library file and JSON config file to your project source.

      The JSON that is used to configure Adobe Mobile has an exclusive key for media heartbeats called `mediaHeartbeat`. This is where the configuration parameters for the media heartbeats belong.

      **Tip:** A sample `ADBMobileConfig` JSON file is provided with the package. Contact your Adobe representatives for the settings.

      For example:

      ```json
      {
        "version":"1.0",
        "analytics":{
          "rsids":"
        },
        "marketingCloud":{
          "org":"
        },
        "target":{
          "clientCode":"
        }
      }
      ```
"timeout":5
},
"audienceManager":{
  "server":"";
},
"acquisition":{
  "server":"example.com",
  "appid":"sample-app-id"
},
"mediaHeartbeat":{
  "server":"example.com",
  "publisher":"sample-publisher",
  "channel":"sample-channel",
  "ssl":false,
  "ovp":"sample-ovp",
  "sdkVersion":"sample-sdk",
  "playerName":"roku"
}

<table>
<thead>
<tr>
<th>Config Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>String that represents the URL of the tracking endpoint on the backend.</td>
</tr>
<tr>
<td>publisher</td>
<td>String that represents the content publisher unique identifier.</td>
</tr>
<tr>
<td>channel</td>
<td>String that represents the name of the content distribution channel.</td>
</tr>
<tr>
<td>ssl</td>
<td>Boolean that represents whether SSL should be used for tracking calls.</td>
</tr>
<tr>
<td>ovp</td>
<td>String that represents the name of the video player provider.</td>
</tr>
<tr>
<td>sdkversion</td>
<td>String that represents the current version of the app/SDK.</td>
</tr>
<tr>
<td>playerName</td>
<td>String that represents the name of the player.</td>
</tr>
</tbody>
</table>

**Important:** If **mediaHeartbeat** is incorrectly configured, the media module (VHL) enters an error state and will stop sending tracking calls.

2. Configure Experience Cloud Visitor ID.

The Experience Cloud Visitor ID service provides a universal Visitor ID across Experience Cloud solutions. The Visitor ID service is required by Video heartbeat and other Marketing Cloud integrations.
Verify that your ADBMobileConfig config contains your marketingCloud organization ID.

```
"marketingCloud": {
    "org": YOUR-MCORG-ID"
}
```

Experience Cloud organization IDs uniquely identify each client company in the Adobe Marketing Cloud and appear similar to the following value: 016D5C175213CCA80A490D05@AdobeOrg.

⚠️ *Important:* Ensure that you include @AdobeOrg.

After the configuration is complete, an Experience Cloud Visitor ID is generated and is included on all hits. Other Visitor IDs, such as custom and automatically-generated, continue to be sent with each hit.

**Experience Cloud Visitor ID Service Methods**

💡 *Tip:* Experience Cloud Visitor ID methods are prefixed with visitor.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visitorMarketingCloudID</td>
<td>Retrieves the Experience Cloud visitor ID from the visitor ID service. ADBMobile().visitorMarketingCloudID()</td>
</tr>
<tr>
<td>visitorSyncIdentifiers</td>
<td>With the Experience Cloud Visitor ID, you can set additional customer IDs that can be associated with each visitor. The Visitor API accepts multiple customer IDs for the same visitor and a customer type identifier to separate the scope of the different customer IDs. This method corresponds to setCustomerIDs in the JavaScript library. For example: identifiers = {} identifiers[&quot;idType&quot;] = &quot;idValue&quot; ADBMobile().visitorSyncIdentifiers(identifiers)</td>
</tr>
<tr>
<td>setAdvertisingIdentifier</td>
<td>Used to set the Roku ID for Advertising (RIDA) on the SDK. For example: ADBMobile().setAdvertisingIdentifier(&quot;&lt;sample_roku_identifier_for_advertising&gt;&quot;) Get the Roku ID for Advertising (RIDA) using the Roku SDK’s getRIDA() API.</td>
</tr>
</tbody>
</table>

**Postbacks** - For more information about configuring postbacks, see [Configure Postbacks](#).
Track audio and video playback

Overview

⚠️ Important: This documentation covers tracking in version 2.x of the SDK. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: Download SDKs

Tracking core playback includes tracking media load, media start, media pause, and media complete. Although not mandatory, tracking buffering and seeking are also core components used for tracking content playback. In your media player API, identify the player events that correspond with the Media SDK tracking calls, and code your event handlers to call tracking APIs, and to populate required and optional variables.

On media load:
- Create the media object
- Populate metadata
- Call `trackSessionStart`; for example: `trackSessionStart(mediaObject, contextData)`

On media start:
- Call `trackPlay`

On pause/resume:
- Call `trackPause`
- Call `trackPlay` when playback resumes

On media complete:
- Call `trackComplete`

On media abort:
- Call `trackSessionEnd`

When scrubbing starts:
- Call `trackEvent(SeekStart)`

When scrubbing ends:
- Call `trackEvent(SeekComplete)`

When buffering starts:
- Call `trackEvent(BufferStart)`;

When buffering ends:
- Call `trackEvent(BufferComplete)`;

💡 Tip: The playhead position is set as part of the set-up and configuration code. For more information about `getCurrentPlayheadTime`, see Set up and configure: Overview.
Implement

1. **Initial tracking setup** - Identify when the user triggers the intention of playback (the user clicks play and/or autoplay is on) and create a `MediaObject` instance using the media information for content name, content ID, content length, and stream type.

   **MediaObject reference:**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Content name</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaid</td>
<td>Content unique identifier</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Content length</td>
<td>Yes</td>
</tr>
<tr>
<td>streamType</td>
<td>Stream type</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaType</td>
<td>Media type (audio or video content)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

   **StreamType constants:**

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOD</td>
<td>Stream type for Video on Demand.</td>
</tr>
<tr>
<td>LIVE</td>
<td>Stream type for Live content.</td>
</tr>
<tr>
<td>LINEAR</td>
<td>Stream type for Linear content.</td>
</tr>
<tr>
<td>AOD</td>
<td>Stream type for audio on demand</td>
</tr>
<tr>
<td>AUDIOBOOK</td>
<td>Stream type for audio book</td>
</tr>
<tr>
<td>PODCAST</td>
<td>Stream type for Podcast</td>
</tr>
</tbody>
</table>

   **MediaType constants:**

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Media type for Audio streams.</td>
</tr>
<tr>
<td>Video</td>
<td>Media type for Video streams.</td>
</tr>
</tbody>
</table>

   The general format for creating the `MediaObject` is: `MediaHeartbeat.createMediaObject(<MEDIA_NAME>, <MEDIA_ID>, <MEDIA_LENGTH>, <STREAM_TYPE>, <MEDIA_TYPE>);`

2. **Attach metadata** - Optionally attach standard and/or custom metadata objects to the tracking session through context data variables.

   • Standard metadata -

     **Note:** Attaching the standard metadata object to the media object is optional.

     Instantiate a standard metadata object, populate the desired variables, and set the metadata object on the Media Heartbeat object.

     See the comprehensive list of metadata here: **Audio and video parameters**
3. **Track the intention to start playback** - To begin tracking a session, call `trackSessionStart` on the Media Heartbeat instance.

   **Important:** `trackSessionStart` tracks the user intention of playback, not the beginning of the playback. This API is used to load the data/metadata and to estimate the time-to-start QoS metric (the time duration between `trackSessionStart` and `trackPlay`).

   **Note:** If you are not using custom metadata, simply send an empty object for the `data` argument in `trackSessionStart`.

4. **Track the actual start of playback** - Identify the event from the media player for the beginning of the playback, where the first frame of the content is rendered on the screen, and call `trackPlay`.

5. **Track the completion of playback** - Identify the event from the media player for the completion of the playback, where the user has watched the content until the end, and call `trackComplete`.

6. **Track the end of the session** - Identify the event from the media player for the unloading/closing of the playback, where the user closes the content and/or the content is completed and has been unloaded, and call `trackSessionEnd`.

   **Important:** `trackSessionEnd` marks the end of a tracking session. If the session was successfully watched to completion, where the user watched the content until the end, ensure that `trackComplete` is called before `trackSessionEnd`. Any other `track*` API call is ignored after `trackSessionEnd`, except for `trackSessionStart` for a new tracking session.

7. **Track all possible pause scenarios** - Identify the event from the media player for pause and call `trackPause`.

   **Pause Scenarios** - Identify any scenario in which the Player will pause and make sure that `trackPause` is properly called. The following scenarios all require that your app call `trackPause()`:

   - The user explicitly hits pause in the app.
   - The player puts itself into the Pause state.
   - *(Mobile Apps)* - The user puts the application into the background, but you want the app to keep the session open.
   - *(Mobile Apps)* - Any type of system interrupt occurs that causes an application to be backgrounded. For example, the user receives a call, or a pop up from another application occurs, but you want the application to keep the session alive to give the user the opportunity to resume the content from the point of interruption.

8. Identify the event from the player for play and/or resume from pause and call `trackPlay`.

   **Tip:** This may be the same event source that was used in Step 4. Ensure that each `trackPause()` API call is paired with a following `trackPlay()` API call when the playback resumes.

9. Listen for playback seeking events from the media player. On seek start event notification, track seeking using the `SeekStart` event.

10. On seek complete notification from the media player, track the end of seeking using the `SeekComplete` event.

11. Listen for the playback buffering events from media player, and on buffer start event notification, track buffering using the `BufferStart` event.
12. On buffer complete notification from the media player, track the end of buffering using the BufferComplete event. See examples of each step in the following platform-specific topics, and look at the sample players included with your SDKs.

For a simple example of playback tracking, see this use of the JavaScript 2.x SDK in an HTML5 player:

```javascript
/* Call on media start */
if (e.type == "play") {
    // Check for start of media
    if (!sessionStarted) {
        /* Set media info */
        /* MediaHeartbeat.createMediaObject(<MEDIA_NAME>,
           <MEDIA_ID>,
           <MEDIA_LENGTH>,
           MediaHeartbeat.StreamType.VOD);*/
        var mediaInfo = MediaHeartbeat.createMediaObject(document.getElementsByTagName('video')[0].getAttribute("name"),
                                                        document.getElementsByTagName('video')[0].getAttribute("id"),
                                                        video.duration,
                                                        MediaHeartbeat.StreamType.VOD);

        /* Set custom context data */
        var customVideoMetadata = {
            isUserLoggedIn: "false",
            tvStation: "Sample TV station",
            programmer: "Sample programmer"
        };

        /* Set standard video metadata */
        var standardVideoMetadata = {};
        standardVideoMetadata[MediaHeartbeat.VideoMetadataKeys.EPISODE] = "Sample Episode";
        mediaInfo.setValue(MediaHeartbeat.MediaObjectKey.StandardVideoMetadata,
                            standardVideoMetadata);

        // Start Session
        this.mediaHeartbeat.trackSessionStart(mediaInfo, customVideoMetadata);

        // Track play
        this.mediaHeartbeat.trackPlay();
        sessionStarted = true;
    } else {
        // Track play for resuming playback
        this.mediaHeartbeat.trackPlay();
    }
}

/* Call on video complete */
if (e.type == "ended") {
    console.log("video ended");
    this.mediaHeartbeat.trackComplete();
    this.mediaHeartbeat.trackSessionEnd();
    sessionStarted = false;
}

/* Call on pause */
if (e.type == "pause") {
    this.mediaHeartbeat.trackPause();
}

/* Call on scrub start */
if (e.type == "seeking") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekStart);
}

/* Call on scrub stop */
```
if (e.type == "seeked") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete);
};

/* Call on buffer start */
if (e.type == "buffering") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferStart);
};

/* Call on buffer complete */
if (e.type == "buffered") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferComplete);
};

Validate

Content Start
On start of a media player, these key calls are sent in the following order:

1. Media analytics start
2. Heartbeat start
3. Heartbeat analytics start

Calls 1 and 2 contain additional metadata variables for both custom and standard.

Content Play
During regular main content playback, Heartbeat calls are sent to the Heartbeat server every ten seconds.

Content Complete
At the 100% point, on content or at a show boundary on a linear stream, a Heartbeat complete call will be sent.

Content Pause
When the player pauses, player pause event calls will be sent every 10 seconds. After pause ends, the play events should resume.

Content Scrub/Seek
On scrubbing of the playhead, no special tracking calls are sent. However, when playback resumes after scrubbing, the playhead value should reflect the new position in the main content.

Content Buffer
When the media player buffers, player buffer event calls are sent every 10 seconds. After buffering ends, the play events should resume.

Track core

Track core playback on Android

⚠️ Important: This documentation covers tracking in version 2.x of the SDK. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guide for Android here: Download SDKs

1. Initial tracking setup - Identify when the user triggers the intention of playback (the user clicks play and/or autoplay is on) and create a MediaObject instance.
**createMediaObject API**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Media name</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaId</td>
<td>Media unique identifier</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Media length</td>
<td>Yes</td>
</tr>
<tr>
<td>streamType</td>
<td>Stream type (see StreamType constants below)</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaType</td>
<td>Media type (see MediaType constants below)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**StreamType constants:**

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOD</td>
<td>Stream type for Video on Demand.</td>
</tr>
<tr>
<td>LIVE</td>
<td>Stream type for Live content.</td>
</tr>
<tr>
<td>LINEAR</td>
<td>Stream type for Linear content.</td>
</tr>
<tr>
<td>AOD</td>
<td>Stream type for Audio On Demand</td>
</tr>
<tr>
<td>AUDIOBOOK</td>
<td>Stream type for Audio Book</td>
</tr>
<tr>
<td>PODCAST</td>
<td>Stream type for Podcast</td>
</tr>
</tbody>
</table>

**MediaType constants:**

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Media type for Audio streams.</td>
</tr>
<tr>
<td>Video</td>
<td>Media type for Video streams.</td>
</tr>
</tbody>
</table>

The general format for creating the media object:

```
MediaHeartbeat.createMediaObject(<MEDIA_NAME>, <MEDIA_ID>, <MEDIA_LENGTH>, <STREAM_TYPE>, <MEDIA_TYPE>);
```

2. **Attach metadata** - Optionally attach standard and/or custom metadata objects to the tracking session through context data variables.

   • **Standard metadata** - *Implement standard metadata on Android*

     ![Note](image) **Note:** Attaching the standard metadata object to the media object is optional.

     • Media metadata keys API Reference - *Standard metadata keys - Android*

     See the comprehensive set of available metadata here: *Audio and video parameters*

   • **Custom metadata** - Create a dictionary for the custom variables and populate with the data for this media. For example:

     ```java
     HashMap<String, String> mediaMetadata =
     new HashMap<String, String>();
     mediaMetadata.put("isUserLoggedIn", "false");
     ```
mediaMetadata.put("tvStation", "Sample TV Station");
mediaMetadata.put("programmer", "Sample programmer");

3. **Track the intention to start playback** - To begin tracking a media session, call `trackSessionStart` on the Media Heartbeat instance. For example:

```java
public void onVideoLoad(Observable observable, Object data) {
    _heartbeat.trackSessionStart(mediaInfo, mediaMetadata);
}
```

   **Tip:** The second value is the custom media metadata object name that you created in step 2.

   **Important:** `trackSessionStart` tracks the user intention of playback, not the beginning of the playback. This API is used to load the media data/metadata and to estimate the time-to-start QoS metric (the time duration between `trackSessionStart` and `trackPlay`).

   **Note:** If you are not using custom media metadata, simply send an empty object for the second argument in `trackSessionStart`.

4. **Track the actual start of playback** - Identify the event from the media player for the beginning of the media playback, where the first frame of the media is rendered on the screen, and call `trackPlay`:

```java
// Media is rendered on the screen) and call trackPlay.
public void onVideoPlay(Observable observable, Object data) {
    _heartbeat.trackPlay();
}
```

5. **Track the completion of playback** - Identify the event from the media player for the completion of the media playback, where the user has watched the content until the end, and call `trackComplete`:

```java
public void onVideoComplete(Observable observable, Object data) {
    _heartbeat.trackComplete();
}
```

6. **Track the end of the session** - Identify the event from the media player for the unloading/closing of the media playback, where the user closes the media and/or the media is completed and has been unloaded, and call `trackSessionEnd`:

```java
// Closes the media and/or the media completed and unloaded,
// and call trackSessionEnd().
public void onMainVideoUnload(Observable observable, Object data) {
    _heartbeat.trackSessionEnd();
}
```

   **Important:** `trackSessionEnd` marks the end of a media tracking session. If the session was successfully watched to completion, where the user watched the content until the end, ensure that `trackComplete` is called before `trackSessionEnd`. Any other `track*` API call is ignored after `trackSessionEnd`, except for `trackSessionStart` for a new media tracking session.

7. **Track all possible pause scenarios** - Identify the event from the media player for media pause and call `trackPause`:

```java
public void onVideoPause(Observable observable, Object data) {
    _heartbeat.trackPause();
}
```
Pause Scenarios - Identify any scenario in which the media player will pause and make sure that `trackPause` is properly called. The following scenarios all require that your app call `trackPause()`:

- The user explicitly hits pause in the app.
- The player puts itself into the Pause state.
- *(Mobile Apps)* - The user puts the application into the background, but you want the app to keep the session open.
- *(Mobile Apps)* - Any type of system interrupt occurs that causes an application to be backgrounded. For example, the user receives a call, or a pop up from another application occurs, but you want the application to keep the session alive to give the user the opportunity to resume the media from the point of interruption.

8. Identify the event from the player for media play and/or media resume from pause and call `trackPlay`

```java
// trackPlay()
public void onVideoPlay(Observable observable, Object data) {
    _heartbeat.trackPlay();
}
```

**Tip:** This may be the same event source that was used in Step 4. Ensure that each `trackPause()` API call is paired with a following `trackPlay()` API call when the media playback resumes.

See the following for additional information on tracking core playback:

- Tracking scenarios: *VOD playback with no ads*
- Sample player included with the Android SDK for a complete tracking example.

**Track core playback on iOS**

**Important:** This documentation covers tracking in version 2.x of the SDK. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: Download SDKs

**Implement**

1. **Initial tracking setup** - Identify when the user triggers the intention of playback (the user clicks play and/or autoplay is on) and create a `MediaObject` instance.

   **createMediaObjectWithName API**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Video name</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaid</td>
<td>Video unique identifier</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Video length</td>
<td>Yes</td>
</tr>
<tr>
<td>streamType</td>
<td>Stream type (see StreamType constants below)</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaType</td>
<td>Media type (Audio</td>
<td>Video )</td>
</tr>
</tbody>
</table>

   **StreamType constants:**
<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBMediaHeartbeatStreamTypeVOD</td>
<td>Stream type for Video on Demand</td>
</tr>
<tr>
<td>ADBMediaHeartbeatStreamTypeLIVE</td>
<td>Stream type for Live content</td>
</tr>
<tr>
<td>ADBMediaHeartbeatStreamTypeLINEAR</td>
<td>Stream type for Linear content</td>
</tr>
<tr>
<td>ADBMediaHeartbeatStreamTypeAOD</td>
<td>Stream type for Audio On Demand</td>
</tr>
<tr>
<td>ADBMediaHeartbeatStreamTypeAUDIOBOOK</td>
<td>Stream type for Audio Book</td>
</tr>
<tr>
<td>ADBMediaHeartbeatStreamTypePODCAST</td>
<td>Stream type for Podcast</td>
</tr>
</tbody>
</table>

**MediaType constants:**

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBMediaTypeAudio</td>
<td>Media type for Audio streams.</td>
</tr>
<tr>
<td>ADBMediaTypeVideo</td>
<td>Media type for Video streams.</td>
</tr>
</tbody>
</table>

The general format for creating the MediaObject:

```c
ADBMediaObject *mediaObject = 
    [ADBMediaHeartbeat createMediaObjectWithName:<MEDIA_NAME>
     mediaId:<MEDIA_ID>
     length:<MEDIA_LENGTH>
     streamType:<STREAM_TYPE>
     mediaType: <MEDIA_TYPE>];
```

2. **Attach video metadata** - Optionally attach standard and/or custom video metadata objects to the video tracking session through context data variables.

   - **Standard video metadata** -
     
     Note: Attaching the standard video metadata object to the media object is optional.

     - Implement standard metadata on iOS
     - Video metadata keys - iOS metadata keys
       - See the comprehensive list of video metadata here: Audio and video parameters

   - **Custom metadata** - Create a variable object for the custom variables and populate with the data for this video.
     For example:

     ```c
     NSMutableArray *videoMetadata = [[NSMutableArray alloc] init];
     [videoMetadata setObject:@"false" forKey:@"isUserLoggedIn"];  
     [videoMetadata setObject:@"Sample TV station" forKey:@"tvStation"];  
     ```

3. **Track the intention to start playback** - To begin tracking a media session, call `trackSessionStart` on the Media Heartbeat instance.

   Tip: The second value is the custom video metadata object name that you created in step 2.

   ```c
   -(void)onMainVideoLoaded:(NSNotification *)notification {  
     // [._mediaHeartbeat trackSessionStart:mediaObject data:nil];  
     [._mediaHeartbeat trackSessionStart:mediaObject data:videoMetadata];  
   }
   ```
**Important:** *trackSessionStart* tracks the user intention of playback, not the beginning of the playback. This API is used to load the video data/metadata and to estimate the time-to-start QoS metric (the time duration between *trackSessionStart* and *trackPlay*).

**Note:** If you are not using custom video metadata, simply send an empty object for the *data* argument in *trackSessionStart*, as shown in the commented out line in the iOS example above.

4. **Track the actual start of playback** - Identify the event from the video player for the beginning of the video playback, where the first frame of the video is rendered on the screen, and call *trackPlay*:

   ```
   -(void)onVideoPlay:(NSNotification *)notification {
       [_mediaHeartbeat trackPlay];
   }
   ```

5. **Track the completion of playback** - Identify the event from the video player for the completion of the video playback, where the user has watched the content until the end, and call *trackComplete*:

   ```
   -(void)onVideoComplete:(NSNotification *)notification {
       [_mediaHeartbeat trackComplete];
   }
   ```

6. **Track the end of the session** - Identify the event from the video player for the unloading/closing of the video playback, where the user closes the video and/or the video is completed and has been unloaded, and call *trackSessionEnd*:

   ```
   -(void)onMainVideoUnloaded:(NSNotification *)notification {
       [_mediaHeartbeat trackSessionEnd];
   }
   ```

   **Important:** *trackSessionEnd* marks the end of a video tracking session. If the session was successfully watched to completion, where the user watched the content until the end, ensure that *trackComplete* is called before *trackSessionEnd*. Any other *track* API call is ignored after *trackSessionEnd*, except for *trackSessionStart* for a new video tracking session.

7. **Track all possible pause scenarios** - Identify the event from the video player for video pause and call *trackPause*:

   ```
   -(void)onVideoPause:(NSNotification *)notification {
       [_mediaHeartbeat trackPause];
   }
   ```

   **Pause Scenarios** - Identify any scenario in which the Video Player will pause and make sure that *trackPause* is properly called. The following scenarios all require that your app call *trackPause()*:

   - The user explicitly hits pause in the app.
   - The player puts itself into the Pause state.
   - *(Mobile Apps)* - The user puts the application into the background, but you want the app to keep the session open.
   - *(Mobile Apps)* - Any type of system interrupt occurs that causes an application to be backgrounded. For example, the user receives a call, or a pop up from another application occurs, but you want the application to keep the session alive to give the user the opportunity to resume the video from the point of interruption.
8. Identify the event from the player for video play and/or video resume from pause and call `trackPlay`:

```objective-c
-(void)onVideoPlay:(NSNotification *)notification {
  [_mediaHeartbeat trackPlay];
}
```

**Tip:** This may be the same event source that was used in Step 4. Ensure that each `trackPause()` API call is paired with a following `trackPlay()` API call when the video playback resumes.

See the following for additional information on tracking core playback:

- Tracking scenarios: **VOD playback with no ads**
- Sample player included with the iOS SDK for a complete tracking example.

**Track core playback on JavaScript**

⚠️ **Important:** This documentation covers tracking in version 2.x of the SDK. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: Download SDKs

Audio tracking requires AppMeasurement library v2.11.0 or higher.

1. **Initial tracking setup** - Identify when the user triggers the intention of playback (the user clicks play and/or autoplay is on) and create a `MediaObject` instance.

**createMediaObject API**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Media name</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaid</td>
<td>Media unique identifier</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Media length</td>
<td>Yes</td>
</tr>
<tr>
<td>streamType</td>
<td>Stream type (see <code>StreamType constants</code> below)</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaType</td>
<td>Media type (see <code>MediaType constants</code> below)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**StreamType constants:**

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOD</td>
<td>Stream type for Video on Demand.</td>
</tr>
<tr>
<td>LIVE</td>
<td>Stream type for LIVE content.</td>
</tr>
<tr>
<td>LINEAR</td>
<td>Stream type for LINEAR content.</td>
</tr>
<tr>
<td>AOD</td>
<td>Stream type for Audio On Demand</td>
</tr>
<tr>
<td>AUDIOBOOK</td>
<td>Stream type for Audio Book</td>
</tr>
<tr>
<td>PODCAST</td>
<td>Stream type for Podcast</td>
</tr>
</tbody>
</table>

**MediaType constants:**
<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Media type for Audio streams.</td>
</tr>
<tr>
<td>Video</td>
<td>Media type for Video streams.</td>
</tr>
</tbody>
</table>

The general format for creating the `MediaObject`:

```javascript
var mediaObject = 
    MediaHeartbeat.createMediaObject(
        <MEDIA_NAME>,
        <MEDIA_ID>,
        <MEDIA_LENGTH>,
        <STREAM_TYPE>,
        <MEDIA_TYPE>);
```

2. **Attach metadata** - Optionally attach standard and/or custom metadata objects to the tracking session through context data variables.

- **Standard metadata** - *Implement standard metadata on JavaScript*

  ![Tip](https://via.placeholder.com/15)

  *Note: Attaching the standard metadata object to the media object is optional.*

- **Media metadata keys API Reference** - *Standard metadata keys - JavaScript*

  See the comprehensive set of available metadata here: *Audio and video parameters*

- **Custom metadata** - Create a variable object for the custom variables and populate with the data for this media. For example:

  ```javascript
  /* Set custom context data */
  var customMediaMetadata = {
      isUserLoggedIn: "false",
      tvStation: "Sample TV station",
      programmer: "Sample programmer"
  };
  ```

3. **Track the intention to start playback** - To begin tracking a media session, call `trackSessionStart` on the Media Heartbeat instance:

```javascript
mediaHeartbeat.trackSessionStart(mediaObject, customMediaMetadata);
```

  ![Tip](https://via.placeholder.com/15)

  *Tip: The second value is the custom media metadata object name that you created in step 2.*

  ![Important](https://via.placeholder.com/15)

  *Important: `trackSessionStart` tracks the user intention of playback, not the beginning of the playback. This API is used to load the media data/metadata and to estimate the time-to-start QoS metric (the time duration between `trackSessionStart` and `trackPlay`).*

  ![Note](https://via.placeholder.com/15)

  *Note: If you are not using custom media metadata, simply send an empty object for the data argument in `trackSessionStart`.*

4. **Track the actual start of playback** - Identify the event from the media player for the beginning of the playback, where the first frame of the media is rendered on the screen, and call `trackPlay`:

```javascript
mediaHeartbeat.trackPlay();
```
5. **Track the completion of playback** - Identify the event from the media player for the completion of the playback, where the user has consumed the content until the end, and call `trackComplete`:

```javascript
mediaHeartbeat.trackComplete();
```

6. **Track the end of the session** - Identify the event from the media player for the unloading/closing of the playback, where the user closes the media and/or the media is completed and has been unloaded, and call `trackSessionEnd`:

```javascript
mediaHeartbeat.trackSessionEnd();
```

**Important:** `trackSessionEnd` marks the end of a tracking session. If the session was successfully watched to completion, where the user watched the content until the end, ensure that `trackComplete` is called before `trackSessionEnd`. Any other `track*` API call is ignored after `trackSessionEnd`, except for `trackSessionStart` for a new tracking session.

7. **Track all possible pause scenarios** - Identify the event from the media player for pause and call `trackPause`:

```javascript
mediaHeartbeat.trackPause();
```

**Pause Scenarios** - Identify any scenario in which the media player will pause and make sure that `trackPause` is properly called. The following scenarios all require that your app call `trackPause()`:

- The user explicitly hits pause in the app.
- The player puts itself into the Pause state.
- (Mobile Apps) - The user puts the application into the background, but you want the app to keep the session open.
- (Mobile Apps) - Any type of system interrupt occurs that causes an application to be backgrounded. For example, the user receives a call, or a pop up from another application occurs, but you want the application to keep the session alive to give the user the opportunity to resume from the point of interruption.

8. Identify the event from the player for play and/or resume from pause and call `trackPlay`:

```javascript
mediaHeartbeat.trackPlay();
```

**Tip:** This may be the same event source that was used in Step 4. Ensure that each `trackPause()` API call is paired with a following `trackPlay()` API call when the playback resumes.

- Tracking scenarios: **VOD playback with no ads**
- Sample player included with the JavaScript SDK for a complete tracking example.

**Track core playback on Chromecast**

**Important:** This documentation covers tracking in version 2.x of the SDK. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: Download SDKs

1. **Initial tracking setup** - Identify when the user triggers the intention of playback (the user clicks play and/or autoplay is on) and create a `MediaObject` instance.

   The general format for creating the media object: `createMediaObject(name, mediaId, length, streamType, mediaType)`
MediaObject reference: createMediaObject

```javascript
mediaObject = ADBMobile.media.createMediaObject(<name>, <id>, <duration>, <streamType>, <mediaType>);
```

**StreamType constants:**
https://adobe-marketing-cloud.github.io/media-sdks/reference/chromecast/ADBMobile.media.html#.StreamType

**MediaType constants:**
https://adobe-marketing-cloud.github.io/media-sdks/reference/chromecast/ADBMobile.media.html#.MediaType

2. **Attach metadata** - Optionally attach standard and/or custom metadata objects to the media tracking session through context data variables.

   - **Standard metadata** - Implement standard metadata on Chromecast

     ![Note](attachment://image.png) Attaching the standard metadata object to the media object is optional.

   - **Custom metadata** - Create a variable object for the custom variables and populate with the data for this content.

     For example:

     ```javascript
     /* Set custom context data */
     var customMetadata = {
       isUserLoggedIn: "false",
       tvStation: "Sample TV station",
       programmer: "Sample programmer"
     };
     ```

3. **Track the intention to start playback** - To begin tracking a media session, call `trackSessionStart` on the media object.

   ![Tip](attachment://image.png) The second value is the custom metadata object name that you created in step 2.

   ```javascript
   ADBMobile.media.trackSessionStart(mediaObject, customMetadata);
   ```

   ![Important](attachment://image.png) `trackSessionStart` tracks the user intention of playback, not the beginning of the playback. This API is used to load the content data/metadata and to estimate the time-to-start QoS metric (the time duration between `trackSessionStart` and `trackPlay`).

   ![Note](attachment://image.png) If you are not using custom metadata, simply send an empty object for the `data` argument in `trackSessionStart`, as shown in the commented out line in the iOS example above.

4. **Track the actual start of playback** - Identify the event from the media player for the beginning of the playback, and call `trackPlay`:

   ```javascript
   ADBMobile.media.trackPlay();
   ```

5. **Track the completion of playback** - Identify the event from the media player for the completion of the media playback, where the user has consumed the content to the end, and call `trackComplete`:

   ```javascript
   ADBMobile.media.trackComplete();
   ```

6. **Track the end of the session** - Identify the event from the media player for the unloading/closing of the playback, where the user closes the media and/or the media is completed and has been unloaded, and call `trackSessionEnd`:

   ```javascript
   ADBMobile.media.trackSessionEnd();
   ```
**Important:** *trackSessionEnd* marks the end of a media tracking session. If the session was successfully watched to completion, where the user watched the content until the end, ensure that *trackComplete* is called before *trackSessionEnd*. Any other *track* API call is ignored after *trackSessionEnd*, except for *trackSessionStart* for a new media tracking session.

7. **Track all possible pause scenarios** - Identify the event from the media player for pause and call *trackPause*:

   ```javascript
   ADBMobile.media.trackPause();
   ```

   **Pause Scenarios** - Identify any scenario in which the media player will pause and make sure that *trackPause* is properly called. The following scenarios all require that your app call *trackPause()*:
   - The user explicitly hits pause in the app.
   - The player puts itself into the Pause state.
   - *(Mobile Apps)* - The user puts the application into the background, but you want the app to keep the session open.
   - *(Mobile Apps)* - Any type of system interrupt occurs that causes an application to be backgrounded. For example, the user receives a call, or a pop up from another application occurs, but you want the application to keep the session alive to give the user the opportunity to resume the content from the point of interruption.

8. Identify the event from the player for play and/or resume from pause and call *trackPlay*:

   ```javascript
   ADBMobile.media.trackPlay();
   ```

   **Tip:** This may be the same event source that was used in Step 4. Ensure that each *trackPause()* API call is paired with a following *trackPlay()* API call when the playback resumes.

   - Tracking scenarios: *VOD playback with no ads*
   - Sample player included with the Chromecast SDK for a complete tracking example.

### Track core playback on Roku

**Important:** This documentation covers tracking in version 2.x of the SDK. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: [Download SDKs](#)

1. **Initial tracking setup** - Identify when the user triggers the intention of playback (the user clicks play and/or autoplay is on) and create a *MediaObject* instance.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Media name</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaid</td>
<td>Media unique identifier</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Media length</td>
<td>Yes</td>
</tr>
<tr>
<td>streamType</td>
<td>Stream type (see <strong>StreamType</strong> constants below)</td>
<td>Yes</td>
</tr>
<tr>
<td>mediaType</td>
<td>Media type (see <strong>MediaType</strong> constants below)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
StreamType constants:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIA_STREAM_TYPE_VOD</td>
<td>Stream type for Video on Demand.</td>
</tr>
<tr>
<td>MEDIA_STREAM_TYPE_LIVE</td>
<td>Stream type for LIVE content.</td>
</tr>
<tr>
<td>MEDIA_STREAM_TYPE_LINEAR</td>
<td>Stream type for LINEAR content.</td>
</tr>
<tr>
<td>MEDIA_STREAM_TYPE_AOD</td>
<td>Stream type for Audio On Demand</td>
</tr>
<tr>
<td>MEDIA_STREAM_TYPE_AUDIOBOOK</td>
<td>Stream type for Audio Book</td>
</tr>
<tr>
<td>MEDIA_STREAM_TYPE_PODCAST</td>
<td>Stream type for Podcast</td>
</tr>
</tbody>
</table>

MediaType constants:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIA_STREAM_TYPE_AUDIO</td>
<td>Media type for Audio streams.</td>
</tr>
<tr>
<td>MEDIA_STREAM_TYPE_VIDEO</td>
<td>Media type for Video streams.</td>
</tr>
</tbody>
</table>

Create a media info object for video with VOD content:

```python
mediaInfo = adb_media_init_mediainfo(
    "<MEDIA_NAME>",
    "<MEDIA_ID>",
    600,
    ADBMobile().MEDIA_STREAM_TYPE_VOD,
    ADBMobile().MEDIA_TYPE_VIDEO
)
```

or

```python
mediaInfo = adb_media_init_mediainfo()
mediaInfo.name = "<MEDIA_NAME>"
mediaInfo.id = "<MEDIA_ID>"
mediaInfo.length = 600
mediaInfo.streamType = ADBMobile().MEDIA_STREAM_TYPE_VOD
mediaInfo.mediaType = ADBMobile().MEDIA_TYPE_VIDEO
```

Create a media info object for video with AOD content:

```python
' Create a media info object
' for audio with AOD content

mediaInfo = adb_media_init_mediainfo(
    "<MEDIA_NAME>",
    "<MEDIA_ID>",
    600,
    ADBMobile().MEDIA_STREAM_TYPE_AOD,
    ADBMobile().MEDIA_TYPE_AUDIO
)
```

or

```python
mediaInfo = adb_media_init_mediainfo()
mediaInfo.name = "<MEDIA_NAME>"
mediaInfo.id = "<MEDIA_ID>"
mediaInfo.length = 600
mediaInfo.streamType = ADBMobile().MEDIA_STREAM_TYPE_AOD
mediaInfo.mediaType = ADBMobile().MEDIA_TYPE_AUDIO
```

• Roku:
2. **Attach metadata** - Optionally attach standard and/or custom metadata objects to the tracking session through context data variables.
   • **Standard metadata** -
     
     *Note:* Attaching the standard metadata object to the media object is optional.

**Implement standard metadata on Roku**

• **Custom metadata** - Create a variable object for the custom variables and populate with the data for this content. For example:

```javascript
mediaContextData = {}
mediaContextData["cmk1"] = "cmv1"
mediaContextData["cmk2"] = "cmv2"
```

3. **Track the intention to start playback** - To begin tracking a media session, call `trackSessionStart` on the Media Heartbeat instance:

   ```javascript
   ADBMobile().mediaTrackSessionStart(mediaInfo,mediaContextData)
   ```

   *Tip:* The second value is the custom metadata object name that you created in step 2.

   *Important:* `trackSessionStart` tracks the user intention of playback, not the beginning of the playback. This API is used to load the data/metadata and to estimate the time-to-start QoS metric (the time duration between `trackSessionStart` and `trackPlay`).

   *Note:* If you are not using custom metadata, simply send an empty object for the data argument in `trackSessionStart`.

4. **Track the actual start of playback** - Identify the event from the media player for the beginning of the playback, and call `trackPlay`:

   ```javascript
   ADBMobile().mediaTrackPlay()
   ```

5. **Track the completion of playback** - Identify the event from the media player for the completion of the playback, where the user has consumed the content to the end, and call `trackComplete`:

   ```javascript
   ADBMobile().mediaTrackComplete()
   ```

6. **Track the end of the session** - Identify the event from the media player for the unloading/closing of the playback, where the user closes the media and/or the content is completed and has been unloaded, and call `trackSessionEnd`:

   ```javascript
   ADBMobile().mediaTrackSessionEnd()
   ```

   *Important:* `trackSessionEnd` marks the end of a tracking session. If the session was successfully watched to completion, where the user watched the content until the end, ensure that `trackComplete` is called before `trackSessionEnd`. Any other `track*` API call is ignored after `trackSessionEnd`, except for `trackSessionStart` for a new tracking session.

7. **Track all possible pause scenarios** - Identify the event from the media player for pause and call `trackPause`:

   ```javascript
   ADBMobile().mediaTrackPause()
   ```
Pause Scenarios - Identify any scenario in which the media player will pause and make sure that `trackPause` is properly called. The following scenarios all require that your app call `trackPause()`:

- The user explicitly hits pause in the app.
- The player puts itself into the Pause state.
- (Mobile Apps) - The user puts the application into the background, but you want the app to keep the session open.
- (Mobile Apps) - Any type of system interrupt occurs that causes an application to be backgrounded. For example, the user receives a call, or a pop up from another application occurs, but you want the application to keep the session alive to give the user the opportunity to resume the content from the point of interruption.

8. Identify the event from the player for play and/or resume from pause and call `trackPlay`:

```java
ADBMobile().mediaTrackPlay()
```

**Tip:** This may be the same event source that was used in Step 4. Ensure that each `trackPause()` API call is paired with a following `trackPlay()` API call when the playback resumes.

- Tracking scenarios: **VOD playback with no ads**
- Sample player included with the Roku SDK for a complete tracking example.

Track buffering

Track buffering on Android

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: Download SDKs.

Buffer tracking constants:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaHeartbeat.Event.BufferStart</td>
<td>Constant for tracking Buffer Start event</td>
</tr>
<tr>
<td>MediaHeartbeat.Event.BufferComplete</td>
<td>Constant for tracking Buffer Complete event</td>
</tr>
</tbody>
</table>

1. Listen for the playback buffering events from media player, and on buffer start event notification, track buffering using the `BufferStart` event:

```java
public void onBufferStart(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.BufferStart, null, null);
}
```

2. On buffer complete notification from the media player, track the end of buffering using the `BufferComplete` event:

```java
public void onBufferComplete(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.BufferComplete, null, null);
}
```

See the tracking scenario **VOD playback with buffering** for more information.
Track buffering on iOS

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

### Buffer tracking constants:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBMediaHeartbeatEventBufferStart</td>
<td>Constant for tracking Buffer Start event</td>
</tr>
<tr>
<td>ADBMediaHeartbeatEventBufferComplete</td>
<td>Constant for tracking Buffer Complete event</td>
</tr>
</tbody>
</table>

### Implement

1. Listen for the playback buffering events from media player, and on buffer start event notification, track buffering using the `BufferStart` event:

   ```swift
   -(void)onBufferStart:(NSNotification *)notification {
       [mediaHeartbeat trackEvent:ADBMediaHeartbeatEventBufferStart
       mediaObject:nil
       data:nil];
   }
   ```

2. On buffer complete notification from the media player, track the end of buffering using the `BufferComplete` event:

   ```swift
   -(void)onBufferComplete:(NSNotification *)notification {
       [mediaHeartbeat trackEvent:ADBMediaHeartbeatEventBufferComplete
       mediaObject:nil
       data:nil];
   }
   ```

See the tracking scenario *VOD playback with buffering* for more information.

Track buffering on JavaScript

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

### Buffer tracking constants:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferStart</td>
<td>Constant for tracking Buffer Start event</td>
</tr>
<tr>
<td>BufferComplete</td>
<td>Constant for tracking Buffer Complete event</td>
</tr>
</tbody>
</table>

### Implement

1. Listen for the playback buffering events from media player, and on buffer start event notification, track buffering using the `BufferStart` event.

   ```javascript
   onBufferStart = function() {
       this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferStart);
   };
   ```
2. On buffer complete notification from the media player, track the end of buffering using the BufferComplete event.

```javascript
_onBufferComplete = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferComplete);
};
```

See the tracking scenario **VOD playback with buffering** for more information.

**Track buffering on Chromecast**

⚠️ **Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

**Buffer tracking constants:**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferStart</td>
<td>Constant for tracking Buffer Start event</td>
</tr>
<tr>
<td>BufferComplete</td>
<td>Constant for tracking Buffer Complete event</td>
</tr>
</tbody>
</table>

**Implement**

1. Listen for the playback buffering events from media player, and on buffer start event notification, track buffering using the BufferStart event: `trackEvent`

   ```javascript
   ADBMobile.media.trackEvent(ADBMobile.media.Event.BufferStart);
   ```

2. On buffer complete notification from the media player, track the end of buffering using the BufferComplete event: `trackEvent`

   ```javascript
   ADBMobile.media.trackEvent(ADBMobile.media.Event.BufferComplete);
   ```

See the tracking scenario **VOD playback with buffering** for more information.

**Track buffering on Roku**

⚠️ **Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

**Buffer tracking constants:**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferStart</td>
<td>Constant for tracking Buffer Start event</td>
</tr>
<tr>
<td>BufferComplete</td>
<td>Constant for tracking Buffer Complete event</td>
</tr>
</tbody>
</table>

**Implement**

1. Listen for the playback buffering events from media player, and on buffer start event notification, track buffering using the BufferStart event.

   ```javascript
   bufferSize = {}
   bufferContextData = {}
   ADBMobile().mediaTrackEvent(MEDIA_BUFFER_START, bufferSize, bufferContextData)
   ```
2. On buffer complete notification from the media player, track the end of buffering using the **BufferComplete** event.

```java
bufferInfo = {}
bufferContextData = {}
ADBMobile().mediaTrackEvent(MEDIA_BUFFER_COMPLETE, bufferInfo, bufferContextData)
```

See the tracking scenario *VOD playback with buffering* for more information.

**Track seeking**

**Track seeking on Android**

⚠️ **Important**: The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

**Seek tracking constants:**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaHeartbeat.Event.SeekStart</td>
<td>Constant for tracking Seek Start event.</td>
</tr>
<tr>
<td>MediaHeartbeat.Event.SeekComplete</td>
<td>Constant for tracking Seek Complete event.</td>
</tr>
</tbody>
</table>

**Implement**

1. Listen for the playback seeking events from the media player, and on seek start event notification, track seeking using the **SeekStart** event:

```java
public void onSeekStart(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.SeekStart, null, null);
}
```

2. On seek complete notification from the media player, track the end of seeking using the **SeekComplete** event:

```java
public void onSeekComplete(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete, null, null);
}
```

See the tracking scenario *VOD playback with seeking in the main content* for more information.

**Track seeking on iOS**

⚠️ **Important**: The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

**Seek tracking constants:**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBMediaHeartbeatEventSeekStart</td>
<td>Constant for tracking Seek Start event.</td>
</tr>
<tr>
<td>ADBMediaHeartbeatEventSeekComplete</td>
<td>Constant for tracking Seek Complete event.</td>
</tr>
</tbody>
</table>

**Implement**
1. Listen for the playback seeking events from the media player, and on seek start event notification, track seeking using the SeekStart event:

```swift
- (void)onSeekStart:(NSNotification *)notification {
    [mediaHeartbeat trackEvent:ADBMediaHeartbeatEventSeekStart
        mediaObject:nil
        data:nil];
}
```

2. On seek complete notification from the media player, track the end of seeking using the SeekComplete event:

```swift
- (void)onSeekComplete:(NSNotification *)notification {
    [mediaHeartbeat trackEvent:ADBMediaHeartbeatEventSeekComplete
        mediaObject:nil
        data:nil];
}
```

See the tracking scenario [VOD playback with seeking in the main content](#) for more information.

### Track seeking on JavaScript

⚠️ **Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

#### Seek tracking constants:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeekStart</td>
<td>Constant for tracking Seek Start event.</td>
</tr>
<tr>
<td>SeekComplete</td>
<td>Constant for tracking Seek Complete event.</td>
</tr>
</tbody>
</table>

#### Implement

1. Listen for the playback seeking events from the media player, and on seek start event notification, track seeking using the SeekStart event:

```javascript
_onSeekStart = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekStart);
};
```

2. On seek complete notification from the media player, track the end of seeking using the SeekComplete event:

```javascript
_onSeekComplete = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete);
};
```

See the tracking scenario [VOD playback with seeking in the main content](#) for more information.

### Track seeking on Chromecast

⚠️ **Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

#### Seek tracking constants:
<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeekStart</td>
<td>Constant for tracking Seek Start event.</td>
</tr>
<tr>
<td>SeekComplete</td>
<td>Constant for tracking Seek Complete event.</td>
</tr>
</tbody>
</table>

**Implement**

1. Listen for the playback seeking events from the media player, and on seek start event notification, track seeking using the `SeekStart` event: `trackEvent`

ADBMobile.media.trackEvent(ADBMobile.media.Event.SeekStart);

2. On seek complete notification from the media player, track the end of seeking using the `SeekComplete` event: `trackEvent`

ADBMobile.media.trackEvent(ADBMobile.media.Event.SeekComplete);

See the tracking scenario *VOD playback with seeking in the main content* for more information.

**Track seeking on Roku**

⚠️ **Important**: The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

**Seek tracking constants:**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeekStart</td>
<td>Constant for tracking Seek Start event.</td>
</tr>
<tr>
<td>SeekComplete</td>
<td>Constant for tracking Seek Complete event.</td>
</tr>
</tbody>
</table>

**Implement**

1. Listen for the playback seeking events from the media player, and on seek start event notification, track seeking using the `SeekStart` event.

   seekContextData = {}  
   seekContextData = {}  
   ADBMobile().mediaTrackEvent(MEDIASEEK_START, seekInfo, seekContextData)

2. On seek complete notification from the media player, track the end of seeking using the `SeekComplete` event.

   seekContextData = {}  
   seekContextData = {}  
   ADBMobile().mediaTrackEvent(MEDIASEEK_COMPLETE, seekInfo, seekContextData)

See the tracking scenario *VOD playback with seeking in the main content* for more information.

**Implement standard metadata**

**Implement standard metadata on Android**

Standard Metadata Constants:
Metadata keys API Reference - Metadata Keys

- Create a HashMap of standard metadata key value pairs using the metadata keys referenced above.
- Set the standard metadata HashMap on MediaInfo using the Standard Metadata constant for the metadata.
- Provide this MediaInfo object while invoking the trackSessionStart() API.

Sample implementation:

```java
// Sample code to set standard metadata for tracking sessions
Map <String, String> standardVideoMetadata= new HashMap<String, String>();
standardVideoMetadata.put(MediaHeartbeat.VideoMetadataKeys.EPISODE, "Sample Episode");
standardVideoMetadata.put(MediaHeartbeat.VideoMetadataKeys.SHOW, "Sample Show");
standardVideoMetadata.put(MediaHeartbeat.VideoMetadataKeys.SEASON, "Sample Season");
mediaInfo.setValue(MediaHeartbeat.MediaObjectKey.StandardVideoMetadata, standardVideoMetadata);
```

Implement standard metadata on iOS

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBMediaObjectKeyStandardMediaMetadata</td>
<td>Constant for attaching standard video metadata on MediaInfo ADBMediaObject</td>
</tr>
</tbody>
</table>

1. Create a dictionary of standard metadata key value pairs using the ADBStandardMetadataKeys (iOS metadata keys).
2. Set the standard metadata dictionary on MediaInfo ADBMediaObject instance using the Standard Metadata constant for metadata.
3. Provide this MediaInfo object while invoking the trackSessionStart API.

```
NSMutableDictionary *standardVideoMetadata =
[[NSMutableDictionary alloc] init];
[standardVideoMetadata setObject:@"Sample Show"
 forKey:ADBVideoMetadataKeySHOW];
[standardVideoMetadata setObject:@"Sample Season"
 forKey:ADBVideoMetadataKeySEASON];
[standardVideoMetadata setObject:@"Sample Episode"
 forKey:ADBVideoMetadataKeyEPISODE];
[mediaObject setValue:standardVideoMetadata
 forKey:ADBMediaObjectKeyStandardMediaMetadata];
```

```
NSMutableDictionary *standardAudioMetadata =
[[NSMutableDictionary alloc] init];
[standardAudioMetadata setObject:@"Sample Album"
 forKey:ADBAudioMetadataKeyALBUM];
[standardAudioMetadata setObject:@"Sample Label"
 forKey:ADBAudioMetadataKeyLABEL];
```
iOS metadata keys

iOS API Reference

From ADBStandardMetadataKeys.h:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBVideoMetadataKeySHOW</td>
<td>Show</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeySEASON</td>
<td>Season</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyEPISODE</td>
<td>Episode</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyASSET_ID</td>
<td>Asset</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyGENRE</td>
<td>Genre</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyFIRST_AIR_DATE</td>
<td>First air date</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyFIRST_DIGITAL_DAT</td>
<td>First digital date</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyRATING</td>
<td>Rating</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyORIGINATOR</td>
<td>Originator</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyNETWORK</td>
<td>Network</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeySHOW_TYPE</td>
<td>Show type</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyAD_LOAD</td>
<td>Ad load</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyMVPD</td>
<td>mvpd</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyAUTHORIZED</td>
<td>Authorization</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyDAY_PART</td>
<td>Day part</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeyFEED</td>
<td>Feed</td>
<td>Video</td>
</tr>
<tr>
<td>ADBVideoMetadataKeySTREAM_FORMAT</td>
<td>Stream format</td>
<td>Video</td>
</tr>
<tr>
<td>ADBAdMetadataKeyADVERTISER</td>
<td>Advertiser</td>
<td>Ad</td>
</tr>
<tr>
<td>ADBAdMetadataKeyCAMPAIGN_ID</td>
<td>Campaign Id</td>
<td>Ad</td>
</tr>
<tr>
<td>ADBAdMetadataKeyCREATIVE_ID</td>
<td>Creative Id</td>
<td>Ad</td>
</tr>
<tr>
<td>ADBAdMetadataKeyPLACEMENT_ID</td>
<td>Placement Id</td>
<td>Ad</td>
</tr>
<tr>
<td>ADBAdMetadataKeySITE_ID</td>
<td>Site Id</td>
<td>Ad</td>
</tr>
<tr>
<td>ADBAdMetadataKeyCREATIVE_URL</td>
<td>Creative Url</td>
<td>Ad</td>
</tr>
</tbody>
</table>
Implement standard metadata on JavaScript

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StandardVideoMetadata</td>
<td>Constant for attaching standard video metadata on Video MediaObject</td>
</tr>
</tbody>
</table>

Instantiate a standard video metadata object, populate the desired variables, and set the metadata object on the Media Heartbeat object. For example:

```javascript
_onVideoLoad = function () {
  //Create the Media Object
  var mediaInfo = MediaHeartbeat.createMediaObject(<MEDIA_NAME>,
  <MEDIA_ID>,
  <MEDIA_LENGTH>,
  <STREAM_TYPE>,
  <MEDIA_TYPE>);

  //Set standard Video Metadata
  var standardVideoMetadata = {};
  standardVideoMetadata[MediaHeartbeat.VideoMetadataKeys.EPISODE] = "Sample Episode";
  mediaInfo.setValue(MediaObjectKey.StandardVideoMetadata, standardVideoMetadata);
  this._mediaHeartbeat.trackSessionStart(mediaInfo, contextData);
};
```

Implement standard metadata on Chromecast

Instantiate a standard video metadata object, populate the desired variables, and set the metadata object on the Media Heartbeat object. For example:

```javascript
var standardVideoMetadata = {};
standardVideoMetadata[VideoMetadataKeys.SHOW] = "Sample show";
standardVideoMetadata[VideoMetadataKeys.SEASON] = "Sample season";
```

See the comprehensive list of video metadata here: [Audio and video parameters](#)

Chromecast metadata keys

Standard video and ad metadata can be set on media and ad info objects respectively. Using the constants keys for video/ad metadata set the dictionary containing standard metadata on info object before calling the track APIs. Refer the tables below for the entire list of standard metadata constants, followed by sample.

Metadata constants

<table>
<thead>
<tr>
<th>Metadata Name</th>
<th>Context Data Key</th>
<th>Constant Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>a.media.show</td>
<td>ADBMobile.media.VideoMetadataKeys.SHOW</td>
</tr>
<tr>
<td>Season</td>
<td>a.media.season</td>
<td>ADBMobile.media.VideoMetadataKeys.SEASON</td>
</tr>
<tr>
<td>Metadata Name</td>
<td>Context Data Key</td>
<td>Constant Name</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>Episode</td>
<td>a.media.episode</td>
<td>ADBMobile.media.VideoMetadataKeys.EPISODE</td>
</tr>
<tr>
<td>Asset</td>
<td>a.media.asset</td>
<td>ADBMobile.media.VideoMetadataKeys.TMS_ID</td>
</tr>
<tr>
<td>Genre</td>
<td>a.media.genre</td>
<td>ADBMobile.media.VideoMetadataKeys.GENRE</td>
</tr>
<tr>
<td>First Air Date</td>
<td>a.media.airDate</td>
<td>ADBMobile.media.VideoMetadataKeys.FIRST_AIR_DATE</td>
</tr>
<tr>
<td>First Digital Air Date</td>
<td>a.media.digitalDate</td>
<td>ADBMobile.media.VideoMetadataKeys.FIRST_DIGITAL_DATE</td>
</tr>
<tr>
<td>Rating</td>
<td>a.media.rating</td>
<td>ADBMobile.media.VideoMetadataKeys.RATING</td>
</tr>
<tr>
<td>Originator</td>
<td>a.media.originator</td>
<td>ADBMobile.media.VideoMetadataKeys.ORIGINATOR</td>
</tr>
<tr>
<td>Network</td>
<td>a.media.network</td>
<td>ADBMobile.media.VideoMetadataKeys.NETWORK</td>
</tr>
<tr>
<td>Show Type</td>
<td>a.media.type</td>
<td>ADBMobile.media.VideoMetadataKeys.SHOW_TYPE</td>
</tr>
<tr>
<td>Ad Load</td>
<td>a.media.adLoad</td>
<td>ADBMobile.media.VideoMetadataKeys.AD_LOAD</td>
</tr>
<tr>
<td>MVPD</td>
<td>a.media.pass.mvpd</td>
<td>ADBMobile.media.VideoMetadataKeys.MVPD</td>
</tr>
<tr>
<td>Authorized</td>
<td>a.media.pass.auth</td>
<td>ADBMobile.media.VideoMetadataKeys.AUTHORIZED</td>
</tr>
<tr>
<td>Day Part</td>
<td>a.media.dayPart</td>
<td>ADBMobile.media.VideoMetadataKeys.DAY_PART</td>
</tr>
<tr>
<td>Feed</td>
<td>a.media.feed</td>
<td>ADBMobile.media.VideoMetadataKeys.FEED</td>
</tr>
<tr>
<td>Stream Format</td>
<td>a.media.format</td>
<td>ADBMobile.media.VideoMetadataKeys.STREAM_FORMAT</td>
</tr>
</tbody>
</table>

**Ad metadata constants**

<table>
<thead>
<tr>
<th>Metadata Name</th>
<th>Context Data Key</th>
<th>Constant Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advertiser</td>
<td>a.media.ad.advertiser</td>
<td>ADBMobile.media.AdMetadataKeys.ADVERTISER</td>
</tr>
<tr>
<td>Campaign ID</td>
<td>a.media.ad.campaign</td>
<td>ADBMobile.media.AdMetadataKeys.CAMPAIGN_ID</td>
</tr>
<tr>
<td>Creative ID</td>
<td>a.media.ad.creative</td>
<td>ADBMobile.media.AdMetadataKeys.CREATIVE_ID</td>
</tr>
<tr>
<td>Placement ID</td>
<td>a.media.ad.placement</td>
<td>ADBMobile.media.AdMetadataKeys.PLACEMENT_ID</td>
</tr>
</tbody>
</table>
### Sample implementation for Chromecast

```javascript
// setting Standard Video Metadata as context data on trackLoad API
mediaContextData = {};
mediaMetadata["videotype"] = "episode";

standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.SHOW] = "sample show";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.SEASON] = "sample season";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.TMS_ID] = "sample tms_id";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.GENRE] = "sample genre";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.FIRST_AIR_DATE] = "sample first_air_date";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.FIRST_DIGITAL_DATE] = "sample first_digital_date";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.RATING] = "sample rating";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.ORIGINATOR] = "sample originator";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.NETWORK] = "sample network";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.SHOW_TYPE] = "sample show type";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.AD_LOAD] = "sample ad load";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.MVPD] = "sample mvpd";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.AUTHORIZED] = "sample authorized";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.DAY_PART] = "sample day_part";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.FEED] = "sample feed";
standardVideoMetadata[ADBMobile.media.VideoMetadataKeys.STREAM_FORMAT] = "sample format";

var mediaObject = ADBMobile.media.createMediaObject(content.name, content.id, content.length, content.streamType);
ADBMobile.media.trackSessionStart(mediaObject, mediaMetadata);
```

### Implement standard metadata on Roku

Instantiate a standard metadata object, populate the desired variables, and set the metadata object on the Media Heartbeat object.

#### Video:

```javascript
standardMetadata = {};
standardMetadata[ADBMobile().MEDIA_VideoMetadataKeySHOW] = "sample show"
standardMetadata[ADBMobile().MEDIA_VideoMetadataKeySEASON] = "sample season"
standardMetadata[ADBMobile().MEDIA_VideoMetadataKeyEPISODE] = "sample episode"

mediaInfo[ADBMobile().MEDIA_STANDARD_MEDIA_METADATA] = standardMetadata
```
Audio:

```python
standardMetadata = {}
standardMetadata[ADBMobile().MEDIA_AudioMetadataKeyARTIST] = "sample artist"
standardMetadata[ADBMobile().MEDIA_AudioMetadataKeyALBUM] = "sample album"
standardMetadata[ADBMobile().MEDIA_AudioMetadataKeyLABEL] = "sample label"

mediaInfo[ADBMobile().MEDIA_STANDARD_MEDIA_METADATA] = standardMetadata
```

See the comprehensive list of video metadata here: Audio and video parameters

Roku metadata keys

Standard video and ad metadata can be set on media and ad info objects respectively. Using the constants keys for video/ad metadata set the dictionary containing standard metadata on info object before calling the track APIs. Refer the tables below for the entire list of standard metadata constants, followed by sample.

### Video metadata constants

<table>
<thead>
<tr>
<th>Metadata Name</th>
<th>Context Data Key</th>
<th>Constant Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>a.media.show</td>
<td>MEDIA_VideoMetadataKeySHOW</td>
</tr>
<tr>
<td>Season</td>
<td>a.media.season</td>
<td>MEDIA_VideoMetadataKeySEASON</td>
</tr>
<tr>
<td>Episode</td>
<td>a.media.episode</td>
<td>MEDIA_VideoMetadataKeyEPISODE</td>
</tr>
<tr>
<td>Asset</td>
<td>a.media.asset</td>
<td>MEDIA_VideoMetadataKeyASSET_ID</td>
</tr>
<tr>
<td>Genre</td>
<td>a.media.genre</td>
<td>MEDIA_VideoMetadataKeyGENRE</td>
</tr>
<tr>
<td>First Air Date</td>
<td>a.media.airDate</td>
<td>MEDIA_VideoMetadataKeyFIRST_AIR_DATE</td>
</tr>
<tr>
<td>First Digital Air Date</td>
<td>a.media.digitalDate</td>
<td>MEDIA_VideoMetadataKeyFIRST_DIGITAL_DATE</td>
</tr>
<tr>
<td>Rating</td>
<td>a.media.rating</td>
<td>MEDIA_VideoMetadataKeyRATING</td>
</tr>
<tr>
<td>Originator</td>
<td>a.media.originator</td>
<td>MEDIA_VideoMetadataKeyORIGINATOR</td>
</tr>
<tr>
<td>Network</td>
<td>a.media.network</td>
<td>MEDIA_VideoMetadataKeyNETWORK</td>
</tr>
<tr>
<td>Show Type</td>
<td>a.media.type</td>
<td>MEDIA_VideoMetadataKeySHOW_TYPE</td>
</tr>
<tr>
<td>Ad Load</td>
<td>a.media.adLoad</td>
<td>MEDIA_VideoMetadataKeyAD_LOAD</td>
</tr>
<tr>
<td>MVPD</td>
<td>a.media.pass.mvpd</td>
<td>MEDIA_VideoMetadataKeyMVPD</td>
</tr>
<tr>
<td>Authorized</td>
<td>a.media.pass.auth</td>
<td>MEDIA_VideoMetadataKeyAUTHORIZED</td>
</tr>
</tbody>
</table>
### Media SDK

#### Metadata Name | Context Data Key | Constant Name
---|---|---
Day Part | a.media.dayPart | MEDIA_VideoMetadataKeyDAY_PART
Feed | a.media.feed | MEDIA_VideoMetadataKeyFEED
Stream Format | a.media.format | MEDIA_VideoMetadataKeySTREAM_FORMAT

#### Audio metadata constants

<table>
<thead>
<tr>
<th>Metadata Name</th>
<th>Context Data Key</th>
<th>Constant Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artist</td>
<td>a.media.artist</td>
<td>MEDIA_AudioMetadataKeyARTIST</td>
</tr>
<tr>
<td>Album</td>
<td>a.media.album</td>
<td>MEDIA_AudioMetadataKeyALBUM</td>
</tr>
<tr>
<td>Label</td>
<td>a.media.label</td>
<td>MEDIA_AudioMetadataKeyLABEL</td>
</tr>
<tr>
<td>Author</td>
<td>a.media.author</td>
<td>MEDIA_AudioMetadataKeyAUTHOR</td>
</tr>
<tr>
<td>Station</td>
<td>a.media.station</td>
<td>MEDIA_AudioMetadataKeySTATION</td>
</tr>
<tr>
<td>Publisher</td>
<td>a.media.publisher</td>
<td>MEDIA_AudioMetadataKeyPUBLISHER</td>
</tr>
</tbody>
</table>

#### Ad metadata constants

<table>
<thead>
<tr>
<th>Metadata Name</th>
<th>Context Data Key</th>
<th>Constant Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advertiser</td>
<td>a.media.ad.advertiser</td>
<td>MEDIA_AdMetadataKeyADVERTISER</td>
</tr>
<tr>
<td>Campaign ID</td>
<td>a.media.ad.campaign</td>
<td>MEDIA_AdMetadataKeyCAMPAIGN_ID</td>
</tr>
<tr>
<td>Creative ID</td>
<td>a.media.ad.creative</td>
<td>MEDIA_AdMetadataKeyCREATIVE_ID</td>
</tr>
<tr>
<td>Placement ID</td>
<td>a.media.ad.placement</td>
<td>MEDIA_AdMetadataKeyPlacement_ID</td>
</tr>
<tr>
<td>Site ID</td>
<td>a.media.ad.site</td>
<td>MEDIA_AdMetadataKeyPlacement_ID</td>
</tr>
<tr>
<td>Creative URL</td>
<td>a.media.ad.creativeURL</td>
<td>MEDIA_AdMetadataKeyCREATIVE_URL</td>
</tr>
</tbody>
</table>

#### Constants

You can use the following constants to track media events:
Table 4: Event Type Constants (Used for the `trackEvent` call)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIA_BUFFER_START</td>
<td>Event Type for Buffer Start</td>
</tr>
<tr>
<td>MEDIA_BUFFER_COMPLETE</td>
<td>Event Type for Buffer Complete</td>
</tr>
<tr>
<td>MEDIA_SEEK_START</td>
<td>Event Type for Seek Start</td>
</tr>
<tr>
<td>MEDIA_SEEK_COMPLETE</td>
<td>Event Type for Seek Complete</td>
</tr>
<tr>
<td>MEDIA_BITRATE_CHANGE</td>
<td>Event Type for Bitrate change</td>
</tr>
<tr>
<td>MEDIACHAPTER_START</td>
<td>Event Type for Chapter Start</td>
</tr>
<tr>
<td>MEDIACHAPTER_COMPLETE</td>
<td>Event Type for Chapter Complete</td>
</tr>
<tr>
<td>MEDIACHAPTER_SKIP</td>
<td>Event Type for Ad Start</td>
</tr>
<tr>
<td>MEDIA_AD_BREAK_START</td>
<td>Event Type for Ad Break</td>
</tr>
<tr>
<td>MEDIA_AD_BREAK_COMPLETE</td>
<td>Event Type for Ad Break Complete</td>
</tr>
<tr>
<td>MEDIA_AD_BREAK_SKIP</td>
<td>Event Type for Ad Break Skip</td>
</tr>
<tr>
<td>MEDIA_AD_START</td>
<td>Event Type for Ad Start</td>
</tr>
<tr>
<td>MEDIA_AD_COMPLETE</td>
<td>Event Type for Ad Complete</td>
</tr>
<tr>
<td>MEDIA_AD_SKIP</td>
<td>Event Type for Ad Skip</td>
</tr>
</tbody>
</table>

Table 5: Content type constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIA_STREAM_TYPE_LIVE</td>
<td>Constant for Stream Type LIVE</td>
</tr>
<tr>
<td>MEDIA_STREAM_TYPE_VOD</td>
<td>Constant for Stream Type VOD</td>
</tr>
</tbody>
</table>

Table 6: MediaObjectKey constants (Used as keys within MediaObject instances)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIA_STANDARD_MEDIA_METADATA</td>
<td>Constant to set video metadata on the MediaInfo object in the trackLoad API.</td>
</tr>
<tr>
<td>MEDIA_STANDARD_AD_METADATA</td>
<td>Constant to set the ad metadata on the EventData object in the trackEvent API for Ad start.</td>
</tr>
<tr>
<td>MEDIA_RESUMED</td>
<td>Constant for sending a video-resumed heartbeat. To resume video tracking of previously stopped content, you need to set the MEDIA_RESUMED property on the mediaInfo object when you call <code>mediaTrackLoad</code>. (MEDIA_RESUMED is not an event that you can track using the <code>mediaTrackEvent</code> API.) MEDIA_RESUMED should be set to</td>
</tr>
</tbody>
</table>
true when an application wants to continue to track content that a user stopped watching but now intends to resume watching.

For example, say a user watches 30% of the content, then closes the app. This will lead to the session being ended. Later, if the same user returns to the same content, and the application allows that user to resume from the same point where they previously left off, then the application should set MEDIA_RESUMED to "true" while calling the mediaTrackLoad API. The result is that these two different media sessions for the same video content can be linked together. Following is the implementation example:

```plaintext
mediaInfo = adb_media_init_mediainfo("test_media_name", "test_media_id", 10, "vod")
mediaInfo[ADBMobile().MEDIA_RESUMED] = true
mediaContextData = {}
ADBMobile().mediaTrackLoad(mediaInfo, mediaContextData)
```

This will create a new session for the video, but it also causes the SDK to send a heartbeat request with the event type "resume", which can be used in reporting to tie two different media sessions together.

### Table 7: Other constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_SOURCE_PLAYER</td>
<td>Constant for Error source being Player</td>
</tr>
</tbody>
</table>

### Track ads

**Overview**

**Important:** The following instructions provide guidance for implementation using the 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: Download SDKs.

Ad playback includes tracking ad breaks, ad starts, ad completes, and ad skips. You can use the media player's API to identify key player events and to populate the required and optional ad variables.

Here are the key elements you use to track ad playback:

**On ad break start, including pre-roll:**

- Create an adBreak object instance for the ad break. For example, adBreakObject.
- Call trackEvent for the ad break start with your adBreakObject.

**On every ad asset start:**
• Create an ad object instance for the ad asset. For example, adObject.
• Populate the ad metadata, adCustomMetadata.
• Call trackEvent for the ad start.

On every ad complete
• Call trackEvent for the ad complete.

On ad skip
• Call trackEvent for the ad skip.

On ad break complete
• Call trackEvent for the ad break complete.

Implement

Ad tracking constants:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdBreakStart</td>
<td>Constant for tracking AdBreak Start event</td>
</tr>
<tr>
<td>AdBreakComplete</td>
<td>Constant for tracking AdBreak Complete event</td>
</tr>
<tr>
<td>AdStart</td>
<td>Constant for tracking Ad Start event</td>
</tr>
<tr>
<td>AdComplete</td>
<td>Constant for tracking Ad Complete event</td>
</tr>
<tr>
<td>AdSkip</td>
<td>Constant for tracking Ad Skip event</td>
</tr>
</tbody>
</table>

1. Identify when the ad break boundary begins, including pre-roll, and create an AdBreakObject by using the ad break information.

AdBreakObject reference:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Ad break name such as pre-roll, mid-roll, and post-roll.</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>The number position of the ad break within the content, starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Playhead value at the start of the ad break.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

2. Call trackEvent() with AdBreakStart in the MediaHeartbeat instance to begin tracking the ad break.
3. Identify when the ad starts and create an AdObject instance using the ad information.

AdObject reference:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Friendly name of the ad.</td>
<td>Yes</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Required</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>adId</td>
<td>Unique identifier for the ad.</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>The number position of the ad within the ad break, starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Ad length</td>
<td>Yes</td>
</tr>
</tbody>
</table>

4. Optionally attach standard and/or ad metadata to the tracking session through context data variables.

   - **Standard ad metadata** - For standard ad metadata, create a dictionary of standard ad metadata key value pairs using the keys for your platform.
   - **Custom ad metadata** - For custom metadata, create a variable object for the custom data variables and populate with the data for the current ad.

5. Call `trackEvent()` with the **AdStart** event in the `MediaHeartbeat` instance to begin tracking the ad playback. Include a reference to your custom metadata variable (or an empty object) as the third parameter in the event call.

6. When the ad playback reaches the end of the ad, call `trackEvent()` with the **AdComplete** event.

7. If ad playback did not complete because the user chose to skip the ad, track the **AdSkip** event.

8. If there are any additional ads within the same **AdBreak**, repeat steps 3 through 7 again.

9. When the ad break is complete, use the **AdBreakComplete** event to track it:

The following sample code utilizes the JavaScript 2.x SDK for an HTML5 video player.

```javascript
/* Call on ad break start */
if (e.type == "ad break start") {
    var adBreakObject = MediaHeartbeat.createAdBreakObject("mid-roll", 2, 500);
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakObject);
};

/* Call on ad start */
if (e.type == "ad start") {
    var adObject = MediaHeartbeat.createAdObject("PepsiOne", "123456ab", 1, 30);
    /* Set custom context data */
    var adCustomMetadata = {
        affiliate:"Sample affiliate",
        campaign:"Sample ad campaign",
        creative:"Sample creative"
    }
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adObject, adCustomMetadata);
};

/* Call on ad complete */
if (e.type == "ad complete") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete);
};

/* Call on ad skip */
if (e.type == "ad skip") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdSkip);
};

/* Call on ad break complete */
if (e.type == "ad break complete") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete);
};
```
Validate

Ad Start
On start of an individual ad playback, three key calls are sent in the following order:

1. Video ad analytics start
2. Heartbeat ad start
3. Heartbeat analytics start

Calls 1 and 2 contain additional metadata variables for both custom and standard.

Ad Play
During ad playback, Heartbeat ad play calls are sent to the Heartbeat server every second.

Ad Complete
At the 100% point of an ad, a Heartbeat ad complete call will be sent.

Ad Skip
When an ad is skipped, no events are sent, so the tracking calls will not include the ad information.

_tip: No unique calls are sent on ad break start and ad break complete._

Track ads on Android

_important: The following instructions provide guidance for implementation using the 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: Download SDKs._

Ad tracking constants:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaHeartbeat.Event.AdBreakStart</td>
<td>Constant for tracking AdBreak Start event</td>
</tr>
<tr>
<td>MediaHeartbeat.Event.AdBreakComplete</td>
<td>Constant for tracking AdBreak Complete event</td>
</tr>
<tr>
<td>MediaHeartbeat.Event.AdStart</td>
<td>Constant for tracking Ad Start event</td>
</tr>
<tr>
<td>MediaHeartbeat.Event.AdComplete</td>
<td>Constant for tracking Ad Complete event</td>
</tr>
<tr>
<td>MediaHeartbeat.Event.AdSkip</td>
<td>Constant for tracking Ad Skip event</td>
</tr>
</tbody>
</table>

1. Identify when the ad break boundary begins, including pre-roll, and create an AdBreakObject by using the ad break information.

_AdBreakObject reference:_

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Ad break name such as pre-roll, mid-roll, and post-roll.</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Required Description

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>The number position of the ad break within the content, starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Playhead value at the start of the ad break.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Ad break object creation:

```java
MediaObject adBreakInfo = 
    MediaHeartbeat.createAdBreakObject(<ADBREAK_NAME>,
    <POSITION>,
    <START_TIME>);
```

2. **Call** `trackEvent()` **with** `AdBreakStart` **in** the **MediaHeartbeat** **instance** to **begin** tracking the ad break:

```java
public void onAdBreakStart(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart,
        adBreakInfo,
        null);
}
```

3. **Identify** when the ad starts and **create** an **AdObject** **instance** using the ad information.

### AdObject reference:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Friendly name of the ad.</td>
<td>Yes</td>
</tr>
<tr>
<td>adId</td>
<td>Unique identifier for the ad.</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>The number position of the ad within the ad break, starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Ad length</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Ad object creation:

```java
MediaObject adInfo = 
    MediaHeartbeat.createAdObject(<AD_NAME>,
    <AD_ID>,
    <POSITION>,
    <LENGTH>);
```

4. **Optionally** **attach** standard and/or ad metadata to the video tracking session through context data variables.

   - **Implement standard ad metadata on Android**
   - **Custom ad metadata** - For custom metadata, create a variable object for the custom data variables and populate with the data for the current ad:

     ```java
     HashMap<String, String> adMetadata = new HashMap<String, String>();
     adMetadata.put("affiliate", "Sample affiliate");
     adMetadata.put("campaign", "Sample ad campaign");
     ```

5. **Call** `trackEvent()` **with** the **AdStart** **event** **in** the **MediaHeartbeat** **instance** to **begin** tracking the ad playback.
Include a reference to your custom metadata variable (or an empty object) as the third parameter in the event call:

```java
public void onAdStart(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.AdStart,
                          adInfo,
                          adMetadata);
}
```

6. When the ad playback reaches the end of the ad, call `trackEvent()` with the `AdComplete` event:

```java
public void onAdComplete(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.AdComplete, null, null);
}
```

7. If ad playback did not complete because the user chose to skip the ad, track the `AdSkip` event:

```java
public void onAdSkip(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.AdSkip, null, null);
}
```

8. If there are any additional ads within the same `AdBreak`, repeat steps 3 through 7 again.

9. When the ad break is complete, use the `AdBreakComplete` event to track:

```java
public void onAdBreakComplete(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete, null, null);
}
```

See the tracking scenario **VOD playback with pre-roll ads** for more information.

### Track ads on iOS

**Important:** The following instructions provide guidance for implementation using the 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: Download SDKs.

**Ad tracking constants:**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBMediaHeartbeatEventAdBreakStart</td>
<td>Constant for tracking AdBreak Start event</td>
</tr>
<tr>
<td>ADBMediaHeartbeatEventAdBreakComplete</td>
<td>Constant for tracking AdBreak Complete event</td>
</tr>
<tr>
<td>ADBMediaHeartbeatEventAdStart</td>
<td>Constant for tracking Ad Start event</td>
</tr>
<tr>
<td>ADBMediaHeartbeatEventAdComplete</td>
<td>Constant for tracking Ad Complete event</td>
</tr>
<tr>
<td>ADBMediaHeartbeatEventAdSkip</td>
<td>Constant for tracking Ad Skip event</td>
</tr>
</tbody>
</table>

1. Identify when the ad break boundary begins, including pre-roll, and create an `AdBreakObject` by using the ad break information.

**AdBreakObject reference:**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Ad break name such as pre-roll, mid-roll, and post-roll.</td>
<td>Yes</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Required</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>position</td>
<td>The number position of the ad break within the content, starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Playhead value at the start of the ad break.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Ad break object creation:**

```objective-c
id adBreakObject = [ADBMediaHeartbeat createAdBreakObjectWithName:[ADBREAK_NAME]
    position:[POSITION]
    startTime:[START_TIME]];
```

2. Call `trackEvent()` with `AdBreakStart` in the `MediaHeartbeat` instance to begin tracking the ad break:

```objective-c
- (void)onAdBreakStart:(NSNotification *)notification {
    [_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakStart
        mediaObject:adBreakObject
        data:nil];
}
```

3. Identify when the ad starts and create an `AdObject` instance using the ad information.

**AdObject reference:**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Friendly name of the ad.</td>
<td>Yes</td>
</tr>
<tr>
<td>adId</td>
<td>Unique identifier for the ad.</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>The number position of the ad within the ad break, starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Ad length</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Ad object creation:**

```objective-c
id adObject = [ADBMediaHeartbeat createAdObjectWithName:[AD_NAME]
    adId:[AD_ID]
    position:[POSITION]
    length:[LENGTH]];
```

4. Optionally attach standard and/or ad metadata to the video tracking session through context data variables.

- **Implement standard ad metadata on iOS**

- **Custom ad metadata** - For custom metadata, create a variable object for the custom data variables and populate with the data for the current ad:

```objective-c
NSMutableDictionary *adDictionary = [[NSMutableDictionary alloc] init];
[adDictionary setObject:@"Sample affiliate" forKey:@"affiliate"];
[adDictionary setObject:@"Sample campaign" forKey:@"campaign"];  
[adDictionary setObject:@"Sample creative" forKey:@"creative"];  
```

5. Call `trackEvent()` with the `AdStart` event in the `MediaHeartbeat` instance to begin tracking the ad playback.

Include a reference to your custom metadata variable (or an empty object) as the third parameter in the event call:

```objective-c
- (void)onAdStart:(NSNotification *)notification {
    [_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdStart
        mediaObject:adObject
        data:adDictionary];
}
```
6. When the ad playback reaches the end of the ad, call `trackEvent()` with the `AdComplete` event.

   ```swift
   - (void)onAdComplete:(NSNotification *)notification {
     [_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdComplete
          mediaObject:nil
          data:nil];
   }
   ```

7. If ad playback did not complete because the user chose to skip the ad, track the `AdSkip` event.

   ```swift
   - (void)onAdSkip:(NSNotification *)notification {
     [_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdSkip
          mediaObject:nil
          data:nil];
   }
   ```

8. If there are any additional ads within the same `AdBreak`, repeat steps 3 through 7 again.

9. When the ad break is complete, use the `AdBreakComplete` event to track:

   ```swift
   - (void)onAdBreakComplete:(NSNotification *)notification {
     [_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakComplete
          mediaObject:nil
          data:nil];
   }
   ```

See the tracking scenario *VOD playback with pre-roll ads* for more information.

### Track ads on JavaScript

⚠️ **Important:** The following instructions provide guidance for implementation using the 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: Download SDKs.

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdBreakStart</td>
<td>Constant for tracking AdBreak Start event</td>
</tr>
<tr>
<td>AdBreakComplete</td>
<td>Constant for tracking AdBreak Complete event</td>
</tr>
<tr>
<td>AdStart</td>
<td>Constant for tracking Ad Start event</td>
</tr>
<tr>
<td>AdComplete</td>
<td>Constant for tracking Ad Complete event</td>
</tr>
<tr>
<td>AdSkip</td>
<td>Constant for tracking Ad Skip event</td>
</tr>
</tbody>
</table>

1. Identify when the ad break boundary begins, including pre-roll, and create an `AdBreakObject` by using the ad break information.

   **AdBreakObject reference:**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Ad break name such as pre-roll, mid-roll, and post-roll.</td>
<td>Yes</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Required</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>position</td>
<td>The number position of the ad break starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Playhead value at the start of the ad break.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Ad break object creation:**

```javascript
var adBreakObject = 
    MediaHeartbeat.createAdBreakObject(<ADBREAK_NAME>,
        <POSITION>,
        <START_TIME>);
```

2. Call `trackEvent()` with `AdBreakStart` in the `MediaHeartbeat` instance to begin tracking the ad break:

```javascript
mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakObject);
```

3. Identify when the ad starts and create an `AdObject` instance using the ad information.

**AdObject reference:**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Friendly name of the ad.</td>
<td>Yes</td>
</tr>
<tr>
<td>adId</td>
<td>Unique identifier for the ad.</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>The number position of the ad within the ad break, starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Ad length</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Ad object creation:**

```javascript
var adObject = 
    MediaHeartbeat.createAdObject(<AD_NAME>,
        <AD_ID>,
        <POSITION>,
        <LENGTH>);
```

4. Optionally attach standard and/or ad metadata to the video tracking session through context data variables.
   - **Implement standard ad metadata on JavaScript**
   - **Custom ad metadata** - For custom metadata, create a variable object for the custom data variables and populate with the data for the current ad:

   ```javascript
   /* Set custom context data */
   var adCustomMetadata = { 
       affiliate: "Sample affiliate",
       campaign: "Sample ad campaign",
       creative: "Sample creative"
   };
   ```

5. Call `trackEvent()` with the `AdStart` event in the `MediaHeartbeat` instance to begin tracking the ad playback.
Include a reference to your custom metadata variable (or an empty object) as the third parameter in the event call:

```javascript
_onAdStart = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, 
adObject, 
adCustomMetadata);
};
```

6. When the ad playback reaches the end of the ad, call `trackEvent()` with the `AdComplete` event:

```javascript
_onAdComplete = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete);
};
```

7. If ad playback did not complete because the user chose to skip the ad, track the `AdSkip` event:

```javascript
_onAdSkip = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdSkip);
};
```

8. If there are any additional ads within the same AdBreak, repeat steps 3 through 7 again.

9. When the ad break is complete, use the `AdBreakComplete` event to track:

```javascript
_onAdBreakComplete = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete);
};
```

See the tracking scenario *VOD playback with pre-roll ads* for more information.

## Track ads on Chromecast

⚠️ **Important:** The following instructions provide guidance for implementation using the 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: [Download SDKs](#).

### Implement

#### Ad tracking constants:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdBreakStart</td>
<td>Constant for tracking AdBreak Start event</td>
</tr>
<tr>
<td>AdBreakComplete</td>
<td>Constant for tracking AdBreak Complete event</td>
</tr>
<tr>
<td>AdStart</td>
<td>Constant for tracking Ad Start event</td>
</tr>
<tr>
<td>AdComplete</td>
<td>Constant for tracking Ad Complete event</td>
</tr>
<tr>
<td>AdSkip</td>
<td>Constant for tracking Ad Skip event</td>
</tr>
</tbody>
</table>

1. Identify when the ad break boundary begins, including pre-roll, and create an `AdBreakObject` by using the ad break information.

   **Ad break object creation:** `createAdBreakObject`

   ```javascript
   adBreakInfo = ADBMobile.media.createAdBreakObject("First Ad-Break", 1, AD_START_POS, playerName);
   ```
2. Call `trackEvent()` with `AdBreakStart` in the `MediaHeartbeat` instance to begin tracking the ad break: `trackEvent` 

ADBMobile.media.trackEvent(ADBMobile.media.Event.AdBreakStart, getAdBreakInfo());

3. Identify when the ad asset starts and create an `AdObject` instance using the ad information. 

    **Ad object creation:** `createAdObject`

    ```java
    adInfo = ADBMobile.media.createAdObject("Sample ad", "001", 1, AD_LENGTH);
    ```

4. Optionally attach standard and/or ad metadata to the video tracking session through context data variables. 

    - **Standard ad metadata** - For standard ad metadata, create a dictionary of standard ad metadata key value pairs using the keys for your platform: 
    - **Custom ad metadata** - For custom metadata, create a variable object for the custom data variables and populate with the data for the current ad asset: 

5. Call `trackEvent()` with the `AdStart` event to begin tracking the ad playback. 

    Include a reference to your custom metadata variable (or an empty object) as the third parameter in the event call: `trackEvent` 

    ```java
    ADBMobile.media.trackEvent(ADBMobile.media.Event.AdStart, getAdInfo(), adContextData);
    ```

6. When the ad asset playback reaches the end of the ad, call `trackEvent()` with the `AdComplete` event: `trackEvent` 

    ```java
    ADBMobile.media.trackEvent(ADBMobile.media.Event.AdComplete);
    ```

7. If there are any additional ads within the same `AdBreak`, repeat steps 3 through 6 again. 

8. When the ad break is complete, use the `AdBreakComplete` event to track: `trackEvent` 

    ```java
    ADBMobile.media.trackEvent(ADBMobile.media.Event.AdBreakComplete, getAdBreakInfo());
    ```

See the tracking scenario **VOD playback with pre-roll ads** for more information.

**Track ads on Roku**

⚠️ **Important:** The following instructions provide guidance for implementation using the 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download 1.x Developers Guides here: [Download SDKs](#).

**Ad tracking constants:**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdBreakStart</td>
<td>Constant for tracking AdBreak Start event</td>
</tr>
<tr>
<td>AdBreakComplete</td>
<td>Constant for tracking AdBreak Complete event</td>
</tr>
<tr>
<td>AdStart</td>
<td>Constant for tracking Ad Start event</td>
</tr>
<tr>
<td>AdComplete</td>
<td>Constant for tracking Ad Complete event</td>
</tr>
<tr>
<td>AdSkip</td>
<td>Constant for tracking Ad Skip event</td>
</tr>
</tbody>
</table>

1. Identify when the ad break boundary begins, including pre-roll, and create an `AdBreakObject` by using the ad break information. 

    **AdBreakObject reference:**
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Ad break name such as pre-roll, mid-roll, and post-roll.</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>The number position of the ad break starting with 1.</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Playhead value at the start of the ad break.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

```javascript
' Create an adbreak info object
adBreakInfo = adb_media_init_adbreakinfo()
adBreakInfo.name = <ADBREAK_NAME>
adBreakInfo.startTime = <START_TIME>
adBreakInfo.position = <POSITION>
```

2. Call `trackEvent()` with `AdBreakStart` in the `MediaHeartbeat` instance to begin tracking the ad break:

```javascript
contextData = {}
ADBMobile().mediaTrackEvent(MEDIA_AD_BREAK_START, adBreakInfo, contextData)
```

3. Identify when the ad asset starts and create an `AdObject` instance using the ad information.

```javascript
adInfo =
ad_media_init_adinfo(ad.title,
ad.id,
ad.position,
ad.duration)
```

4. Optionally attach standard and/or ad metadata to the video tracking session through context data variables.
   - **Implement standard ad metadata on Roku**
   - **Custom ad metadata** - For custom metadata, create a variable object for the custom data variables and populate with the data for the current ad asset:

```javascript
contextData = {}
contextData["adinfo1"] = "adinfo2"
contextData["adinfo2"] = "adinfo2"
```

5. Call `trackEvent()` with the `AdStart` event in the `MediaHeartbeat` instance to begin tracking the ad playback:

```javascript
ADBMobile().mediaTrackEvent(ADBMobile().MEDIA_AD_START, adInfo, contextData)
```

6. When the ad asset playback reaches the end of the ad, call `trackEvent()` with the `AdComplete` event.

```javascript
standardAdMetadata = {}
contextData = {}
ADBMobile().mediaTrackEvent(ADBMobile().MEDIA_AD_COMPLETE, adInfo, contextData)
```

7. If ad playback did not complete because the user chose to skip the ad, track the `AdSkip` event:

```javascript
contextData = {}
ADBMobile().mediaTrackEvent(ADBMobile().MEDIA_AD_SKIP, adInfo, contextData)
```

8. If there are any additional ads within the same `AdBreak`, repeat steps 3 through 7 again.

9. When the ad break is complete, use the `AdBreakComplete` event to track:

```javascript
contextData = {}
ADBMobile().mediaTrackEvent(MEDIA_AD_BREAK_COMPLETE, adBreakInfo, contextData)
```
See the tracking scenario *VOD playback with pre-roll ads* for more information.

**Implement standard ad metadata**

**Implement standard ad metadata on Android**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
</table>

**Standard ad metadata** - For standard ad metadata, create a dictionary of standard ad metadata key value pairs using the keys for your platform:

```java
// Setting standard Ad Metadata
Map <String, String> standardAdMetadata = new HashMap<String, String>();
standardAdMetadata.put(MediaHeartbeat.AdMetadataKeys.ADVERTISER, "Sample Advertiser");
standardAdMetadata.put(MediaHeartbeat.AdMetadataKeys.CAMPAIGN_ID, "Sample Campaign");
adInfo.setValue(MediaHeartbeat.MediaObjectKey.StandardAdMetadata, standardAdMetadata);
```

**Implement standard ad metadata on iOS**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBMediaObjectKeyStandardAdMetadata</td>
<td>Constant for attaching standard ad metadata on AdInfo ADBMediaObject</td>
</tr>
</tbody>
</table>

**Standard ad metadata** - For standard ad metadata, create a dictionary of standard ad metadata key value pairs using the keys for your platform:

```objective-c
// Sample implementation for using standard metadata keys for Ad
NSMutableDictionary *standardAdMetadata = [[NSMutableDictionary alloc] init];
[standardAdMetadata setObject:@"Sample Advertiser" forKey:ADBAdMetadataKeyADVERTISER];
[standardAdMetadata setObject:@"Sample Campaign" forKey:ADBAdMetadataKeyCAMPAIGN_ID];
[adObject setValue:standardAdMetadata forKey:ADBMediaObjectKeyStandardAdMetadata];
```

**iOS metadata keys**

**Implement standard ad metadata on JavaScript**

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StandardAdMetadata</td>
<td>Constant for attaching standard ad metadata on Ad Object</td>
</tr>
</tbody>
</table>

• **Standard ad metadata** - For standard ad metadata, create a dictionary of standard ad metadata key value pairs using the keys for your platform:

```javascript
var adObject =
    MediaHeartbeat.createAdObject(<AD_NAME>,
        <AD_ID>,
        <POSITION>,
        <LENGTH>);

// Set standard Ad Metadata
```

**Media SDK**
var standardAdMetadata = {};  
standardAdMetadata[MediaHeartbeat.AdMetadataKeys.CAMPAIGN_ID] = "Sample Campaign";  
adObject.setValue(MediaObjectKey.StandardAdMetadata, standardAdMetadata);

Implement standard ad metadata on Roku

**Standard ad metadata** - For standard ad metadata, create a dictionary of standard ad metadata key value pairs using the keys for your platform:

standardAdMetadata = {}  
standardAdMetadata[ADBMobile().MEDIA_AdMetadataKeyCAMPAIGN_ID] = "sample campaign"  
standardAdMetadata[ADBMobile().MEDIA_AdMetadataKeyADVERTISER] = "sample advertiser"  
adInfo[ADBMobile().MEDIA_STANDARD_AD_METADATA] = standardAdMetadata

Track chapters and segments

**Overview**

⚠️ *Important:* The following instructions provide guidance for implementation using 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the Developers Guide here: Download SDKs.

Chapter and segment tracking is available for custom-defined video chapters or segments. Some common uses for chapter tracking are to define custom segments based on video content, such as baseball innings, or to define content segments between ad breaks. Chapter tracking is not required for core video heartbeat implementations.

Chapter tracking includes chapter starts, chapter completes, and chapter skips. You can use the video player API with customized segmentation logic to identify chapter events and to populate the required and optional chapter variables. Here are the key elements of tracking chapter playback:

**On chapter start:**

- Create the chapter object instance for the chapter, `chapterObject`
- Populate the chapter metadata, `chapterCustomMetadata`
- Call `trackEvent(MediaHeartbeat.Event.ChapterStart, chapterObject, chapterCustomMetadata);`

**On chapter complete:**

- Call `trackEvent(MediaHeartbeat.Event.ChapterComplete);`

**On chapter skip:**

- Call `trackEvent(MediaHeartbeat.Event.ChapterSkip);`

**Implement**

1. Identify when the chapter start event occurs and create the `ChapterObject` instance by using the chapter information.

Here is the `ChapterObject` chapter tracking reference:
2. If you include custom metadata for the chapter, create the context data variables for the metadata.

3. To begin tracking the chapter playback, call the `ChapterStart` event in the `MediaHeartbeat` instance.

4. When playback reaches the chapter end boundary, as defined by your custom code, call the `ChapterComplete` event in the `MediaHeartbeat` instance.

5. If chapter playback did not complete because the user chose to skip the chapter (for example, if the user seeks out of the chapter boundary), call the `ChapterSkip` event in the `MediaHeartbeat` instance.

6. If there are any additional chapters, repeat steps 1 through 5.

The following sample code uses the JavaScript 2.x SDK for an HTML5 video player. You should use this code with the core video playback code.

```javascript
/* Call on chapter start */
if (e.type == "chapter start") {
    var chapterObject = MediaHeartbeat.createChapterObject("Inning 5", 5, 500, 2500);
    var chapterCustomMetadata = {
        segmentType: "Baseball Innings",
        segmentName: "Inning 5",
        segmentInfo: "Game Six"
    }
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart, chapterObject, chapterCustomMetadata);
}

/* Call on chapter complete */
if (e.type == "chapter complete") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterComplete);
}

/* Call on chapter skip */
if (e.type == "chapter skip") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterSkip);
}
```

**Validate**

**Chapter Start**

On start of an individual chapter playback, one key calls are sent:

- Heartbeat chapter start*

  *This call contains additional chapter metadata variables.*
Chapter Complete
At the chapter boundary end, a Heartbeat chapter complete call will be sent.

Chapter Skip
When a chapter is skipped, a Heartbeat chapter skip call will be sent.

Track chapters and segments on Android

⚠️ **Important:** The following instructions provide guidance for implementation using 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the Developers Guide here: [Download SDKs](#).

1. Identify when the chapter start event occurs and create the `ChapterObject` instance by using the chapter information.

   Here is the `ChapterObject` chapter tracking reference:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Chapter name</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>Chapter position</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Chapter length</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Chapter start time</td>
<td>Yes</td>
</tr>
</tbody>
</table>

   Note: These variables are only required if you are planning to track chapters.

   Chapter object:
   ```java
   MediaObject chapterDataInfo = 
   MediaHeartbeat.createChapterObject(<CHAPTER_NAME>,
   <POSITION>,
   <LENGTH>,
   <START_TIME>);
   ```

2. If you include custom metadata for the chapter, create the context data variables for the metadata:

   ```java
   HashMap<String, String> chapterMetadata = 
   new HashMap<String, String>();
   chapterMetadata.put("segmentType", "Sample Segment Type");
   chapterMetadata.put("segmentName", "Sample Segment Name");
   chapterMetadata.put("segmentInfo", "Sample Segment Info");
   ```

3. To begin tracking the chapter playback, call the `ChapterStart` event in the `MediaHeartbeat` instance:

   ```java
   public void onChapterStart(Observable observable, Object data) {
   _heartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart,
   chapterDataInfo,
   chapterMetadata);
   }
   ```
4. When playback reaches the chapter end boundary, as defined by your custom code, call the `ChapterComplete` event in the `MediaHeartbeat` instance:

```java
public void onChapterComplete(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.ChapterComplete, null, null);
}
```

5. If chapter playback did not complete because the user chose to skip the chapter (for example, if the user seeks out of the chapter boundary), call the `ChapterSkip` event in the `MediaHeartbeat` instance:

```java
public void onChapterSkip(Observable observable, Object data) {
    _heartbeat.trackEvent(MediaHeartbeat.Event.ChapterSkip, null, null);
}
```

6. If there are any additional chapters, repeat steps 1 through 5.

**Track chapters and segments on iOS**

⚠️ **Important:** The following instructions provide guidance for implementation using 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the Developers Guide here: Download SDks.

1. Identify when the chapter start event occurs and create the `ChapterObject` instance by using the chapter information.

Here is the `ChapterObject` chapter tracking reference:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Chapter name</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>Chapter position</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Chapter length</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Chapter start time</td>
<td>Yes</td>
</tr>
</tbody>
</table>

💡 **Note:** These variables are only required if you are planning to track chapters.

**Chapter object:**

```objective-c
id chapterObject =
    [ADBMediaHeartbeat createChapterObjectWithName:@"CHAPTER_NAME"
        position:@"POSITION"
        length:@"LENGTH"
        startTime:@"START_TIME"];```

2. If you include custom metadata for the chapter, create the context data variables for the metadata:

```objective-c
NSMutableDictionary *chapterDictionary = [[NSMutableDictionary alloc] init];
[chapterDictionary setObject:@"Sample segment type" forKey:@"segmentType"];
[chapterDictionary setObject:@"Sample segment name" forKey:@"segmentName"];```

3. To begin tracking the chapter playback, call the `ChapterStart` event in the `MediaHeartbeat` instance:

```objective-c
-(void)onChapterStart:(NSNotification *)notification {
    _mediaHeartbeat.trackEvent:ADBMediaHeartbeatEventChapterStart```
4. When playback reaches the chapter end boundary, as defined by your custom code, call the `ChapterComplete` event in the `MediaHeartbeat` instance:

   ```swift
   - (void)onChapterComplete:(NSNotification *)notification {
     [_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventChapterComplete
      mediaObject:nil
      data:nil];
   }
   ```

5. If chapter playback did not complete because the user chose to skip the chapter (for example, if the user seeks out of the chapter boundary), call the `ChapterSkip` event in the `MediaHeartbeat` instance:

   ```swift
   - (void)onChapterSkip:(NSNotification *)notification {
     [_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventChapterSkip
      mediaObject:nil
      data:nil];
   }
   ```

6. If there are any additional chapters, repeat steps 1 through 5.

**Track chapters and segments on JavaScript**

> **Important:** The following instructions provide guidance for implementation using 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the Developers Guide here: Download SDKs.

1. Identify when the chapter start event occurs and create the `ChapterObject` instance by using the chapter information.

   Here is the `ChapterObject` chapter tracking reference:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Chapter name</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>Chapter position</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Chapter length</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Chapter start time</td>
<td>Yes</td>
</tr>
</tbody>
</table>

> **Note:** These variables are only required if you are planning to track chapters.

**Chapter object:**

```javascript
var chapterInfo = MediaHeartbeat.createChapterObject(<CHAPTER_NAME>,
   <POSITION>,
   <LENGTH>,
   <START_TIME>);
```
2. If you include custom metadata for the chapter, create the context data variables for the metadata:

```javascript
var chapterCustomMetadata = {
    segmentType: "Sample segment type",
    segmentName: "Sample segment name",
    segmentInfo: "Sample segment info"
};
```

3. To begin tracking the chapter playback, call the `ChapterStart` event in the `MediaHeartbeat` instance:

```javascript
_onChapterStart = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart, chapterObject, chapterCustomMetadata);
};
```

4. When playback reaches the chapter end boundary, as defined by your custom code, call the `ChapterComplete` event in the `MediaHeartbeat` instance:

```javascript
_onChapterComplete = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterComplete);
};
```

5. If chapter playback did not complete because the user chose to skip the chapter (for example, if the user seeks out of the chapter boundary), call the `ChapterSkip` event in the `MediaHeartbeat` instance:

```javascript
_onChapterSkip = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterSkip);
};
```

6. If there are any additional chapters, repeat steps 1 through 5.

**Track chapters and segments on Chromecast**

**Important:** The following instructions provide guidance for implementation using 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the Developers Guide here: Download SDKs.

1. Identify when the chapter start event occurs and create the `ChapterObject` instance by using the chapter information.

   Here is the `ChapterObject` chapter tracking reference:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Chapter name</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>Chapter position</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Chapter length</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Chapter start time</td>
<td>Yes</td>
</tr>
</tbody>
</table>

   **Note:** These variables are only required if you are planning to track chapters.
Chapter object: `createChapterObject`

```javascript
chapterInfo = ADBMobile.media.createChapterObject("First Chapter", 1, CHAPTER1_LENGTH, CHAPTER1_START_POS);
```

2. If you include custom metadata for the chapter, create the context data variables for the metadata:

```javascript
var chapterContextData = {
    segmentType: "Sample segment type"
};
```

3. To begin tracking the chapter playback, track the `ChapterStart` event: `trackEvent`

```javascript
ADBMobile.media.trackEvent(ADBMobile.media.Event.ChapterStart, ChapterInfo, chapterContextData);
```

4. When playback reaches the chapter end boundary, as defined by your custom code, call the `ChapterComplete` event in the `MediaHeartbeat` instance: `trackEvent`

```javascript
ADBMobile.media.trackEvent(ADBMobile.media.Event.ChapterComplete);
```

5. If chapter playback did not complete because the user chose to skip the chapter (for example, if the user seeks out of the chapter boundary), track the `ChapterSkip` event: `trackEvent`

```javascript
ADBMobile.media.trackEvent(ADBMobile.media.Event.ChapterSkip);
```

6. If there are any additional chapters, repeat steps 1 through 5.

**Track chapters and segments on Roku**

⚠️ **Important:** The following instructions provide guidance for implementation using 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the Developers Guide here: [Download SDKs](#).

1. Identify when the chapter start event occurs and create the `ChapterObject` instance by using the chapter information.

   Here is the `ChapterObject` chapter tracking reference:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Chapter name</td>
<td>Yes</td>
</tr>
<tr>
<td>position</td>
<td>Chapter position</td>
<td>Yes</td>
</tr>
<tr>
<td>length</td>
<td>Chapter length</td>
<td>Yes</td>
</tr>
<tr>
<td>startTime</td>
<td>Chapter start time</td>
<td>Yes</td>
</tr>
</tbody>
</table>

   🌟 **Note:** These variables are only required if you are planning to track chapters.

Chapter object:

```javascript
chapterInfo = adb_media_init_chapterinfo(<CHAPTER_NAME>, <POSITION>, <LENGTH>, <START_TIME>);
```
2. If you include custom metadata for the chapter, create the context data variables for the metadata:

```javascript
chapterContextData = {}
chapterContextData["seg_type"] = "seg_type"
chapterContextData["seg_name"] = "seg_name"
chapterContextData["seg_info"] = "seg_info"
```

3. To begin tracking the chapter playback, call the `ChapterStart` event in the `MediaHeartbeat` instance:

```javascript
ADBMobile().mediaTrackEvent(MEDIA_CHAPTER_START, chapterInfo, chapterContextData)
```

4. When playback reaches the chapter end boundary, as defined by your custom code, call the `ChapterComplete` event in the `MediaHeartbeat` instance.

```javascript
chapterContextData = {}
ADBMobile().mediaTrackEvent(MEDIA_CHAPTER_COMPLETE, chapterInfo, chapterContextData)
```

5. If chapter playback did not complete because the user chose to skip the chapter (for example, if the user seeks out of the chapter boundary), call the `ChapterSkip` event in the `MediaHeartbeat` instance.

```javascript
chapterContextData = {}
ADBMobile().mediaTrackEvent(MEDIA_CHAPTER_SKIP, chapterInfo, chapterContextData)
```

6. If there are any additional chapters, repeat steps 1 through 5.

### Track quality of experience

#### Overview

⚠️ **Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: Download SDKs.

Quality of experience tracking includes quality of service (QoS) and error tracking, both are optional elements and are not required for core video heartbeat implementations. You can use the video player API to identify the variables related to QoS and error tracking. Here are the key elements of tracking quality of experience:

- **On any QoS metric changes:**
  - Create or update the QoS object instance for the playback.

- **On all bitrate change events:**
  - Call `trackEvent(Media.Heartbeat.Event.BitrateChange);`

#### Implement

1. Identify when any of QOS metrics change during video playback, create the `MediaObject` using the QoS information, and update the new QoS information.

#### QoSObject variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate</td>
<td>Current bitrate</td>
<td>Yes</td>
</tr>
<tr>
<td>startupTime</td>
<td>Startup time</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Tip: These variables are only required if you are planning to track QoS.

2. Make sure that `getQoSObject()` method returns the most updated QoS information.
3. When playback switches bitrates, call the `BitrateChange` event in the Media Heartbeat instance.

Important: Update the QoS object and call the bitrate change event on every bitrate change. This provides the most accurate QoS data.

The following sample code uses the JavaScript 2.x SDK for an HTML5 video player. You should use this code with the core video playback code.

```javascript
var mediaDelegate = new MediaHeartbeatDelegate();
...

// This is called periodically by MediaHeartbeat instance
mediaDelegate.prototype.getQoSObject = function() {
    return this.qosInfo;
};

if (e.type == "qos_update") {
    var qosInfo = MediaHeartbeat.createQoSObject(<BITRATE>,<STARTUP_TIME>,<FPS>,<DROPPED_FRAMES>);
    mediaDelegate.qosInfo = qosInfo;
};

if (e.type == "bitrate_change") {
    this.mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BitrateChange, qosObject);
};
```

Validate

Bitrate change

On each bitrate change, a Heartbeat `bitrate_change` call will be sent.

Error

On player error, a Heartbeat error call will be sent with the error value included.

Track quality of experience on Android

Important: The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

1. Identify when the bitrate changes during video playback and create the `MediaObject` instance using the QoS information.

QoSObject variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>fps</td>
<td>FPS value</td>
<td>Yes</td>
</tr>
<tr>
<td>droppedFrames</td>
<td>Number of dropped frames</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Tip:** These variables are only required if you are planning to track QoS.

### QoS object creation:

```java
MediaObject qosObject = MediaHeartbeat.createQoSObject(<BITRATE>,
<STARTUP_TIME>,
<FPS>,
<DROPPED_FRAMES>);
```

2. Make sure that `getQoSObject()` method returns the most updated QoS information.

3. When playback switches bitrates, call the `BitrateChange` event in the Media Heartbeat instance:

   ```java
   public void onBitrateChange(Observable observable, Object data) {
       _heartbeat.trackEvent(MediaHeartbeat.Event.BitrateChange, null, null);
   }
   ```

   **Important:** Update the QoS object and call the bitrate change event on every bitrate change. This provides the most accurate QoS data.

### Track quality of experience on iOS

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: Download SDKs.

1. Identify when the bitrate changes during video playback and create the `MediaObject` instance using the QoS information.

### QoSObject variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate</td>
<td>Current bitrate</td>
<td>Yes</td>
</tr>
<tr>
<td>startupTime</td>
<td>Startup time</td>
<td>Yes</td>
</tr>
<tr>
<td>fps</td>
<td>FPS value</td>
<td>Yes</td>
</tr>
<tr>
<td>droppedFrames</td>
<td>Number of dropped frames</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate</td>
<td>Current bitrate</td>
<td>Yes</td>
</tr>
<tr>
<td>startupTime</td>
<td>Startup time</td>
<td>Yes</td>
</tr>
<tr>
<td>fps</td>
<td>FPS value</td>
<td>Yes</td>
</tr>
<tr>
<td>droppedFrames</td>
<td>Number of dropped frames</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Tip:** These variables are only required if you are planning to track QoS.

**QoS object creation:**

```swift
id qosObject = [ADBMediaHeartbeat createQoSObjectWithBitrate:[BITRATE]
startupTime:[STARTUP_TIME]
fps:[FPS]
droppedFrames:[DROPPED_FRAMES]];
```

2. Make sure that `getQoSObject` method returns the most updated QoS information.

3. When playback switches bitrates, call the `BitrateChange` event in the Media Heartbeat instance:

```swift
- (void)onBitrateChange:(NSNotification *)notification {
    [_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventBitrateChange
     mediaObject:nil
data:nil];
}
```

**Important:** Update the QoS object and call the bitrate change event on every bitrate change. This provides the most accurate QoS data.

### Track quality of experience on JavaScript

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

1. Identify when the bitrate changes during video playback and create the `MediaObject` instance using the QoS information.

#### QoSObject variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate</td>
<td>Current bitrate</td>
<td>Yes</td>
</tr>
<tr>
<td>startupTime</td>
<td>Startup time</td>
<td>Yes</td>
</tr>
<tr>
<td>fps</td>
<td>FPS value</td>
<td>Yes</td>
</tr>
<tr>
<td>droppedFrames</td>
<td>Number of dropped frames</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Tip:** These variables are only required if you are planning to track QoS.

**QoS object creation:**

```javascript
// Replace <bitrate>, <startupTime>, <fps> and<br>
// <droppedFrames> with the current playback QoS values.
var qosObject = MediaHeartbeat.createQoSObject(<bitrate>,
                 <startupTime>,
                 <fps>,
                 <droppedFrames>);
```
2. When playback switches bitrates, call the `BitrateChange` event in the Media Heartbeat instance:

```javascript
_onBitrateChange = function() {
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BitrateChange, qosObject);
};
```

**Important:** Update the QoS object and call the bitrate change event on every bitrate change. This provides the most accurate QoS data.

3. Make sure that `getQoSObject()` method returns the most updated QoS information.

4. When the video player encounters an error, and the error event is available to the player API, use `trackError()` to capture the error information. (See `Track errors`.)

**Tip:** Tracking video player errors will not stop the video tracking session. If the video player error prevents the playback from continuing, make sure that the video tracking session is closed by calling `trackSessionEnd()` after calling `trackError()`.

### Track quality of experience on Chromecast

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: Download SDKs.

**Overview**

Quality of experience tracking includes quality of service (QoS) and error tracking, both are optional elements and are not required for core video heartbeat implementations. You can use the video player API to identify the variables related to QoS and error tracking. Here are the key elements of tracking quality of experience:

**Tip:** Additional details for each section is available in the Implement section.

**On all bitrate change events:**

- Create/update the QoS object instance for the playback, `qosObject`
- Call `trackEvent(Media.Heartbeat.Event.BitrateChange, qosObject);`

**On player errors:**

- Call `trackError("video error id");`

**Implement**

1. Identify when the bitrate changes during video playback and create the `MediaObject` instance using the QoS information.

**QoSObject variables:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate</td>
<td>Current bitrate</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Tip:** These variables are only required if you are planning to track QoS.

{| Variable | Description | Required |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>startupTime</td>
<td>Startup time</td>
<td>Yes</td>
</tr>
<tr>
<td>fps</td>
<td>FPS value</td>
<td>Yes</td>
</tr>
<tr>
<td>droppedFrames</td>
<td>Number of dropped frames</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**QoS object creation:** `createQoSObject`

```javascript
qosInfo = ADBMobile.media.createQoSObject(50000, 0, 24, 10);
```

2. When playback switches bitrates, call the `BitrateChange` event in the Media Heartbeat instance: `trackEvent`

```javascript
ADBMobile.media.trackEvent(ADBMobile.media.Event.BitrateChange);
```

**Important:** Update the QoS object and call the bitrate change event on every bitrate change. This provides the most accurate QoS data.

3. Make sure that `getQoSObject()` method returns the most updated QoS information.
4. When the video player encounters an error, and the error event is available to the player API, use `trackError()` to capture the error information. (See **Track errors**.)

**Tip:** Tracking video player errors will not stop the video tracking session. If the video player error prevents the playback from continuing, make sure that the video tracking session is closed by calling `trackSessionEnd()` after calling `trackError()`.

**Track quality of experience on Roku**

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

1. Identify when the bitrate changes during media playback, and use the `mediaUpdateQoS` API to update the QoS info on the Media SDK.

**QoSObject variables:**

**Tip:** These variables are only required if you are planning to track QoS.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate</td>
<td>Current bitrate</td>
<td>Yes</td>
</tr>
<tr>
<td>startupTime</td>
<td>Startup time</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Track errors

**Overview**

⚠️ **Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

#### Implement

1. Track video player errors.

   On error events, call `trackError` with the error information.

   ✨ **Note:** Tracking video player errors will not stop the video tracking session. If the video player error prevents the playback from continuing, make sure that the video tracking session is closed by calling `trackSessionEnd() after calling trackError`. 

---

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>fps</td>
<td>FPS value</td>
<td>Yes</td>
</tr>
<tr>
<td>droppedFrames</td>
<td>Number of dropped frames</td>
<td>Yes</td>
</tr>
</tbody>
</table>

For example:

```
bitrate = 200000
fps = 0
droppedFrames = 1
startupTime = 2
qosinfo = adb_media_init_qosinfo(bitrate, startupTime, fps, droppedFrames)
ADBMobile().mediaUpdateQoS(qosinfo)
```

2. When playback switches bitrates, call `trackEvent(BitrateChange)` to notify the Media SDK that the bitrate changed.

   `ADBMobile().trackMediaEvent(ADBMobile().MEDIA_BITRATE_CHANGE)`

   ✨ **Note:** You need to call `updateQoSObject` with the updated bitrate value.

3. When the media player encounters an error, and the error event is available to the player API, use `trackError()` to capture the error information. (See [Track errors](#).)

   ✨ **Tip:** Tracking media player errors will not stop the media tracking session. If the media player error prevents the playback from continuing, make sure that the media tracking session is closed by calling `trackSessionEnd() after calling trackError`. 

---
Track errors on Android

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: Download SDKs.

1. Track video player errors:

```java
public void onPlayerError(Observable observable, Object data) {
    _heartbeat.trackError("videoErrorID");
}
```

**Note:** Tracking video player errors will not stop the video tracking session. If the video player error prevents the playback from continuing, make sure that the video tracking session is closed by calling `trackSessionEnd` after calling `trackError`.

Track errors on iOS

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: Download SDKs.

Implement

1. Track video player errors:

```swift
-(void)onPlayerError:(NSNotification *)notification {
    [_mediaHeartbeat trackError:@"videoErrorId"];
}
```

**Note:** Tracking video player errors will not stop the video tracking session. If the video player error prevents the playback from continuing, make sure that the video tracking session is closed by calling `trackSessionEnd` after calling `trackError`.

Track errors on JavaScript

**Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: Download SDKs.

Implement

1. Track video player errors:

```javascript
onPlayerError = function() {
    this._mediaHeartbeat.trackError("videoErrorId");
};
```

**Note:** Tracking video player errors will not stop the video tracking session. If the video player error prevents the playback from continuing, make sure that the video tracking session is closed by calling `trackSessionEnd` after calling `trackError`. 
Track errors on Chromecast

⚠️ **Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

**Implement**

1. Track video player errors: `trackError`
   ```javascript
   trackError(errorId)
   ```

  💡 **Note:** Tracking video player errors will not stop the video tracking session. If the video player error prevents the playback from continuing, make sure that the video tracking session is closed by calling `trackSessionEnd` after calling `trackError`.

Track errors on Roku

⚠️ **Important:** The following instructions provide guidance for implementation across all 2.x SDKs. If you are implementing a 1.x version of the SDK, you can download the 1.x Developers Guides here: [Download SDKs](#).

**Implement**

1. Track video player errors:
   ```javascript
   ADBMobile().mediaTrackError(msg.GetMessage(),
   ADBMobile().ERROR_SOURCE_PLAYER)
   ```

  💡 **Note:** Tracking video player errors will not stop the video tracking session. If the video player error prevents the playback from continuing, make sure that the video tracking session is closed by calling `trackSessionEnd` after calling `trackError`.

Opt-out and privacy

**Opt-out / Opt-in**

You can control whether tracking activity is allowed on a specific device:

- **Mobile Apps** - The VA library respects the AdobeMobile library’s privacy and opt-out settings. To opt-out of tracking, you need to use the AdobeMobile library. For more information on the AdobeMobile library’s opt-out and privacy settings, see [Opt-Out and Privacy Settings](#).

- **JavaScript/Browser Apps** - The VA library respects the VisitorAPI privacy and optout settings. To optout of tracking, you need to opt out from the Visitor API service. For further information on optout and privacy, see [Experience Cloud ID Service](#).

- **OTT Apps (Chromecast, Roku)** -

  The OTT SDKs provide General Data Protection Regulation (GDPR)-ready APIs that allow you to set opt status flags for data collection and transmission, and to retrieve locally stored identities.
Note: Media heartbeat tracking calls are also disabled if the privacy status is set to opt-out.

You can control whether or not Analytics data is sent on a specific device using the following settings:

• The `privacyDefault` setting in the `ADBMobile.json` config file. This controls the initial setting and persists until it is changed in code.
• The `ADBMobile().setPrivacyStatus()` method.

• Opt out:
  • Chromecast:
    `ADBMobile.config.setPrivacyStatus(ADBMobile.config.PRIVACY_STATUS_OPT_OUT)`
  • Roku:
    `ADBMobile().setPrivacyStatus(ADBMobile().PRIVACY_STATUS_OPT_OUT)`

Important: When a user opts out of tracking, all of the persisted device data and IDs will be purged until the user opts back in.

• Opt back in:
  • Chromecast:
    `ADBMobile.config.setPrivacyStatus(ADBMobile.config.PRIVACY_STATUS_OPT_IN)`
  • Roku:
    `ADBMobile().setPrivacyStatus(ADBMobile().PRIVACY_STATUS_OPT_IN)`

• Return the current setting:
  • Chromecast:
    `ADBMobile.config.getPrivacyStatus()`
  • Roku:
    `ADBMobile().getPrivacyStatus()`

After the privacy setting is changed using `setPrivacyStatus`, the change is permanent until it is changed again using this method, or the app is uninstalled and reinstalled.

Retrieving Stored Identifiers (OTT Apps)

This information helps you retrieve locally stored user identities from your Roku app.

Important: The method for retrieving all identifiers gets all user identities known and persisted by the SDK. You must call this method before a user opts-out.

The locally stored identities are returned in a JSON string, which might contain:

• Company Context - IMS Org IDs
• User IDs
• Experience Cloud ID (MCID)
• Data Source IDs (DPID, DPUUID)
• Analytics IDs (AVID, AID, VID, and associated RSIDs)
• Audience Manager ID (UUID)

For example:

• **Chromecast:**
  
  ```javascript
  ADBMobile.config.getAllIdentifiersAsync(callback)
  ```

• **Roku:**
  
  ```javascript
  vids = ADBMobile().getAllIdentifiers()
  ```

### Tracking scenarios

#### VOD playback with no ads

**Scenario**

This scenario has one VOD asset, with no ads, which is played once from beginning to end.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks Play</td>
<td><code>trackStart</code></td>
<td>Analytics Content Start, Heartbeat Content Start</td>
<td>This can be either a user clicking Play or an auto-play event.</td>
</tr>
<tr>
<td>First frame of the video</td>
<td><code>trackPlay</code></td>
<td>Heartbeat Content Play</td>
<td>This method triggers the timer, and from this point forward, heartbeats will be sent every 10 seconds for the duration of the playback.</td>
</tr>
<tr>
<td>Content plays</td>
<td></td>
<td>Content Heartbeats</td>
<td></td>
</tr>
<tr>
<td>Content is complete</td>
<td><code>trackComplete</code></td>
<td>Heartbeat Content Complete</td>
<td><em>Complete</em> means that the end of the playhead was reached.</td>
</tr>
</tbody>
</table>

**Parameters**

Many of the same values that you see on Heartbeat Content Start Calls are also seen on Adobe Analytics Content Start Calls. There are many parameters that Adobe uses to populate the various video reports, but only the most important parameters are listed in the following table:

### Table 8: Heartbeat Content Start

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s:sc:rsid</code></td>
<td><code>&lt;Your Adobe Report Suite ID&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>s:sc:tracking_server</code></td>
<td><code>&lt;Your Analytics Tracking Server URL&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>s:user:mid</code></td>
<td>must be set</td>
<td>Should match the mid value on the Adobe Analytics Content Start call.</td>
</tr>
<tr>
<td><code>s:event:type</code></td>
<td>&quot;start&quot;</td>
<td></td>
</tr>
<tr>
<td><code>s:asset:type</code></td>
<td>&quot;main&quot;</td>
<td></td>
</tr>
</tbody>
</table>
### Heartbeat Content Play

These parameters should look nearly identical to the Heartbeat Content Start call, but the key difference is the `s:event:type` parameter. All of the other parameters should still exist.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:asset:video_id</td>
<td><code>&lt;Your Video Name&gt;</code></td>
<td></td>
</tr>
<tr>
<td>s:meta:*</td>
<td>optional</td>
<td>Custom metadata that is set on the video.</td>
</tr>
<tr>
<td>s:event:type</td>
<td>&quot;play&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;main&quot;</td>
<td></td>
</tr>
</tbody>
</table>

### Content heartbeats

During video playback, a timer sends at least one heartbeat every 10 seconds. These heartbeats contain information about playback, ads, buffering, and so on. The exact content of each heartbeat is beyond the scope of this document, but the critical issue is that heartbeats are triggered consistently while playback continues.

In the content heartbeats, look for the following parameters:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;play&quot;</td>
<td></td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>&lt;playhead position&gt; eg.50,60,70</td>
<td>This parameter reflects the current position of the playhead.</td>
</tr>
</tbody>
</table>

### Heartbeat Content Complete

When playback has completed, which means that the end of the playhead is reached, a Heartbeat Content Complete call is sent. This call looks like other Heartbeat calls, but it contains some specific parameters:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;complete&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;main&quot;</td>
<td></td>
</tr>
</tbody>
</table>

### Sample Code

In this scenario, the content is 40 seconds long. It is played until the end without any interruptions.
**Android -**

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_VAL_1, CUSTOM_KEY_1);
videoMetadata.put(CUSTOM_VAL_2, CUSTOM_KEY_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay
//    is used, i.e., there's an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// 2. Call trackPlay() when the playback actually starts,
//    i.e., the first frame of video is rendered on the screen.
_mediaHeartbeat.trackPlay();

// 3. Call trackComplete() when the playback reaches the end,
//    i.e., when the video completes and finishes playing.
_mediaHeartbeat.trackComplete();

// 4. Call trackSessionEnd() when the playback session is over.
//    This method must be called even if the user does not watch
//    the video to completion.
_mediaHeartbeat.trackSessionEnd();
```

**iOS -**

```objective-c
when the user clicks Play
ADBMediaObject *mediaObject =
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
        length:MEDIA_LENGTH
        streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
```
[videoContextData setObject: CUSTOM_VAL_1 forKey: CUSTOM_KEY_1];
[videoContextData setObject: CUSTOM_VAL_2 forKey: CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
//    i.e., there's an intent to start playback.
[_mediaHeartbeat trackSessionStart: mediaObject data: videoContextData];
......
......

// 2. Call trackPlay when the playback actually starts, i.e., when the
//    first frame of main content is rendered on the screen.
[_mediaHeartbeat trackPlay];
......
......

// 3. Call trackComplete when the playback reaches the end, i.e.,
//    when the video completes and finishes playing.
[_mediaHeartbeat trackComplete];
......
......

// 4. Call trackSessionEnd when the playback session is over. This method
//    must be called even if the user does not watch the video to completion.
[_mediaHeartbeat trackSessionEnd];
......
......

**JavaScript**

```
var mediaInfo = MediaHeartbeat.createMediaObject(Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH, MediaHeartbeat.StreamType.VOD);
var videoMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when the user clicks play, or when autoplay is used,
//    i.e., there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);
......
......

// 2. Call trackPlay() when the main content starts, i.e.,
//    the first frame of the video content is rendered on the screen.
this._mediaHeartbeat.trackPlay();

......
......

// 3. Call trackComplete() when the playback reaches the end,
//    i.e., the video completes and finishes playing.
this._mediaHeartbeat.trackComplete();

......
......

// 4. Call trackSessionEnd() when the playback session is over.
//    This method must be called even if the user does not
//    watch the video to completion.
this._mediaHeartbeat.trackSessionEnd();

......
......```
VOD playback with pre-roll ads

In this scenario, pre-roll ads have been inserted before the main content. Unless specified, the network calls are the same as the calls in the **VOD playback with no ads** scenario. The network calls happen at the same time, but the payload is different.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user clicks Play</td>
<td>trackSessionStart</td>
<td>Analytics Content Start, Heartbeat Content Start</td>
<td>The measurement library does not know that there is a pre-roll ad, so these network calls are still identical to the <strong>VOD playback with no ads</strong> scenario.</td>
</tr>
<tr>
<td>The ad starts.</td>
<td></td>
<td>Analytics Ad Start, Heartbeat Ad Start</td>
<td></td>
</tr>
<tr>
<td>The frame of ad #1 is played.</td>
<td>trackPlay</td>
<td>Heartbeat Ad Play</td>
<td>The ad content plays before main content, and the heartbeats start when the ad starts.</td>
</tr>
<tr>
<td>The ad is played.</td>
<td></td>
<td>Ad Heartbeats</td>
<td></td>
</tr>
<tr>
<td>Ad #2 completes playing.</td>
<td>trackEvent:trackAdComplete</td>
<td>Heartbeat Ad Complete</td>
<td>The end of the ad is reached.</td>
</tr>
<tr>
<td>The first frame of ad #2 is played.</td>
<td>trackEvent:AdStart</td>
<td>Analytics Ad Start, Heartbeat Ad Start</td>
<td></td>
</tr>
<tr>
<td>The ad plays.</td>
<td></td>
<td>Ad Heartbeats</td>
<td></td>
</tr>
<tr>
<td>Ad #2 completes playing.</td>
<td>*trackEvent:trackAdComplete, *trackEvent:AdBreakComplete</td>
<td>Heartbeat Ad Complete</td>
<td>The end of the ad and the pod is reached.</td>
</tr>
<tr>
<td>The content plays.</td>
<td></td>
<td>Content Heartbeats</td>
<td>This network call is identical to the <strong>VOD playback with no ads</strong> scenario.</td>
</tr>
<tr>
<td>The content is complete.</td>
<td>trackComplete</td>
<td>Heartbeat Content Complete</td>
<td>This network call is identical to the <strong>VOD playback with no ads</strong> scenario.</td>
</tr>
<tr>
<td>The session is over</td>
<td>trackSessionEnd</td>
<td></td>
<td>SessionEnd means the end of a viewing session. This API must be called even if the user does not watch the video to completion.</td>
</tr>
</tbody>
</table>
Parameters

When ad playback begins, a Heartbeat Ad Start call is sent. If the beginning of the ad does not coincide with the 10-second timer, the Heartbeat Ad Start call is delayed by a few seconds, and the call goes to the next 10-second interval. When this happens, a Content Heartbeat goes out in the same interval, and you can differentiate between the two calls by looking at the event type and the asset type:

Table 9: Heartbeat Ad Start

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;start&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;ad&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Ads follow the same basic model as Content Heartbeats, so the Ad Play call is similar to the Content Play call.

Table 10: Heartbeat Ad Play Call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;play&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;ad&quot;</td>
<td></td>
</tr>
</tbody>
</table>

These parameters are similar to the Content Heartbeats call, but the Ad Heartbeats call contains a few extra parameters:

Table 11: Ad Heartbeats

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;play&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;ad&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:ad_id</td>
<td>&lt;ad ID&gt;</td>
<td></td>
</tr>
<tr>
<td>s:asset:pod_id</td>
<td>&lt;ad pod ID&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Similar to Heartbeat Content Complete calls, when ad playback has completed, and the end of the playhead is reached, a Heartbeat Ad Complete call is sent. This call looks like other Heartbeat Ad calls but contains a couple specific things:

Table 12: Heartbeat Ad Complete Call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;complete&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;ad&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Sample code for a pre-roll ad break

In this scenario, the VOD consists of a pre-roll ad, a second pre-roll ad, and then the content is played.
To view this scenario in Android, set up the following code:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
//    i.e., there's an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// Pre-roll
MediaObject adBreakInfo =
    MediaHeartbeat.createAdBreakObject(ADBREAK_NAME,
    ADBREAK_POSITION,
    ADBREAK_START_TIME);

MediaObject adInfo =
    MediaHeartbeat.createAdObject(AD_NAME,
    AD_ID,
    AD_POSITION,
    AD_LENGTH);

// Context ad data
HashMap<String, String> adMetadata = new HashMap<String, String>();
adMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
adMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 2. Track the MediaHeartbeat.Event.AdBreakStart event when the pre-roll pod starts
// to play. Note that since this is a pre-roll, call must track the
// "MediaHeartbeat.Event.AdBreakStart" event before you call trackPlay().
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo, null);

// 3. Track the MediaHeartbeat.Event.AdStart event when the pre-roll pod's ad starts
// to play. Note that since this is a pre-roll, you must track the
// "MediaHeartbeat.Event.AdStart" event before you call trackPlay().
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);
```
// 4. Call trackPlay() when the playback actually starts, i.e., when the first frame
// of the ad video is rendered on the screen.
_mediaHeartbeat.trackPlay();

........
........

// 5. Track the MediaHeartbeat.Event.AdComplete event when the ad reaches the end,
// i.e., when the ad completes and finishes playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete, null, null);

........
........

// 6. Track the MediaHeartbeat.Event.AdStart event when the pre-roll pod's second ad
// starts to play.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

........
........

// 7. Track the MediaHeartbeat.Event.AdComplete event when the second ad reaches the
// end, i.e., the second ad completes and finishes playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete, null, null);

........
........

// 8. Track the MediaHeartbeat.Event.AdBreakComplete event when all of the ads in the
// pod finish playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete, null, null);

........
........

// 9. Call trackComplete() when the playback reaches the end, i.e., when the video
// completes and finishes playing.
_mediaHeartbeat.trackComplete();

........
........

// 10. Call trackSessionEnd() when the playback session is over. This method must be
// called even if the user does not watch the video to completion.
_mediaHeartbeat.trackSessionEnd();

........
........

• iOS -

To view this scenario in iOS, set up the following code:

// Set up mediaObject
ADBMediaObject *mediaObject =
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
     length:MEDIA_LENGTH
     streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
// i.e., there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

........
........
Media SDK

// Pre-roll
ADBMediaObject *adBreakInfo =
[ADBMediaHeartbeat createAdBreakObjectWithName:AD_BREAK_NAME
position:AD_BREAK_POSITION
startTime:AD_BREAK_START_TIME];
ADBMediaObject *adInfo =
[ADBMediaHeartbeat createAdObjectWithName:AD_NAME
adId:AD_ID
position:AD_POSITION
length:AD_LENGTH];
// context ad data
NSMutableDictionary *adDictionary = [[NSMutableDictionary alloc] init];
[adDictionary setObject:@"custom-val1" forKey:@"custom-key1"];
[adDictionary setObject:@"custom-val2" forKey:@"custom-key2"];
// 2. Track the ADBMediaHeartbeatEventAdBreakStart event when the pre-roll pod
//
starts to play. Note that since this is a pre-roll, you must track the
//
"ADBMediaHeartbeatEventAdBreakStart" event before you call trackPlay.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakStart
mediaObject:adBreakObject
data:nil];
.......
.......
// 3. Track the ADBMediaHeartbeatEventAdStart event when the pre-roll pod's
//
ad starts to play. Note that since this is a pre-roll, you must track
//
the "ADBMediaHeartbeatEventAdStart" event before you call trackPlay.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdStart
mediaObject:adObject
data:adDictionary];
.......
.......
// 4. Call trackPlay when the playback actually starts, i.e., when the
//
first frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];
.......
.......
// 5. Track the ADBMediaHeartbeatEventAdComplete event when the ad reaches
//
the end, i.e., when the video completes and finishes playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdComplete
mediaObject:nil
data:nil];
.......
.......
// 6. Track the ADBMediaHeartbeatEventAdStart event when the pre-roll pod's
//
second ad starts to play.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdStart
mediaObject:adBreakObject
data:nil];
.......
.......
// 7. Track the ADBMediaHeartbeatEventAdComplete event when the second ad
//
reaches the end, i.e., it completes and finishes playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdComplete
mediaObject:nil
data:nil];
.......
.......
// 8. Track the ADBMediaHeartbeatEventAdBreakComplete event when all the
//
ads in the pod finish playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakComplete
mediaObject:adBreakObject

134


data:nil];

// 9. Call trackComplete when the playback reaches the end, i.e., when the
// video completes and finishes playing.
[_mediaHeartbeat trackComplete];

// 10. Call trackSessionEnd when the playback session is over. This method
// must be called even if the user does not watch the video to completion.
[_mediaHeartbeat trackSessionEnd];

• JavaScript -
To view this scenario in JavaScript, enter the following text:

// Set up mediaObject
var mediaInfo =
    MediaHeartbeat.createMediaObject(Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,MediaHeartbeat.StreamType.VOD);

var videoMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay is used,
// i.e., there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// Preroll
var adBreakInfo =
    MediaHeartbeat.createAdBreakObject(ADBREAK_NAME, ADBREAK_POSITION, ADBREAK_START_TIME);
var adInfo =
    MediaHeartbeat.createAdObject(AD_NAME, AD_ID, AD_POSITION, AD_LENGTH);

// Custom ad metadata
var adMetadata = {
    CUSTOM_AD_KEY_1 : CUSTOM_AD_VAL_1,
    CUSTOM_AD_KEY_2 : CUSTOM_AD_VAL_2
};

// 2. Track the MediaHeartbeat.Event.AdBreakStart event when the preroll pod starts to play.
// Note that since this is a preroll, track the MediaHeartbeat.Event.AdBreakStart
// event before you call trackPlay().
this._mediaheartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo, null);

// 3. Track the MediaHeartbeat.Event.AdStart event when the preroll pod's ad starts to play.
// Note that since this is a preroll, track the MediaHeartbeat.Event.AdStart event before
// you call trackPlay().
this._mediaheartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);
// 4. Call trackPlay() when the playback actually starts, i.e., the first frame of the main content is rendered on the screen.
this._mediaHeartbeat.trackPlay();

// 5. Track event MediaHeartbeat.Event.AdComplete when the ad reaches the end, i.e., when it completes and finishes playing.
this._mediaheartbeat.trackEvent(MediaHeartbeat.Event.AdComplete);

// 6. Track the MediaHeartbeat.Event.AdStart event when the preroll pod's second ad starts to play.
this._mediaheartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

// 7. Track the MediaHeartbeat.Event.AdComplete event when the second ad reaches the end, i.e., when it completes and finishes playing.
this._mediaheartbeat.trackEvent(MediaHeartbeat.Event.AdComplete);

// 8. Track the MediaHeartbeat.Event.AdBreakComplete event when all the ads in the pod finish playing.
this._mediaheartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete);

// 9. Call trackComplete() when the playback reaches the end, i.e., when it completes and finishes playing.
this._mediaHeartbeat.trackComplete();

// 10. Call trackSessionEnd() when the playback session is over. This method must be called even if the user does not watch the video to completion.
this._mediaHeartbeat.trackSessionEnd();

Sample code for multiple ad breaks

In this scenario, VOD content is played back with a pre-roll ad, the content, a mid-roll ad, the content, and a post-roll ad.

- Android -
To view this scenario in Android, set up the following code:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
//   i.e., there's an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// Pre-roll
MediaObject adBreakInfo =
    MediaHeartbeat.createAdBreakObject(ADBREAK_NAME,
    ADBREAK_POSITION,
    ADBREAK_START_TIME);
MediaObject adInfo = MediaHeartbeat.createAdObject(AD_NAME,
    AD_ID,
    AD_POSITION,
    AD_LENGTH);

HashMap<String, String> adMetadata = new HashMap<String, String>();
adMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
adMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 2. Track the MediaHeartbeat.Event.AdBreakStart event when the pre-roll pod
//    starts to play. Note that since this is a pre-roll, you must track the
//    "MediaHeartbeat.Event.AdBreakStart" event before you call trackPlay().
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo, null);

// 3. Track the MediaHeartbeat.Event.AdStart event when the pre-roll pod's ad
//    starts to play. Note that since this is a pre-roll, you must track the
//    "MediaHeartbeat.Event.AdStart" event before you call trackPlay().
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

// 4. Call trackPlay() when the playback actually starts, i.e., when the first
//    frame of the main content is rendered on the screen.
_mediaHeartbeat.trackPlay();

// 5. Track the MediaHeartbeat.Event.AdComplete event when the ad reaches the end,
//    i.e., when the ad completes and finishes playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete, null, null);

// 6. Track the MediaHeartbeat.Event.AdBreakComplete event when all of the ads in
//    the pod finish playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete, null, null);
```
// Mid-roll
MediaObject adBreakInfo =
    MediaHeartbeat.createAdBreakObject(mid-roll_BREAK_NAME,
        mid-roll_BREAK_POSITION,
        mid-roll_BREAK_START_TIME);
MediaObject adInfo =
    MediaHeartbeat.createAdObject(mid-roll_AD_NAME,
        mid-roll_AD_ID,
        mid-roll_AD_POSITION,
        mid-roll_AD_LENGTH);

// Context ad data
HashMap<String, String> adMetadata = new HashMap<String, String>();
adMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
adMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 7. Track the MediaHeartbeat.Event.AdBreakStart event when the mid-roll pod
// starts to play.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo, null);

// 8. Track the MediaHeartbeat.Event.AdStart event when the mid-roll pod's ad
// starts to play.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

// 9. Track the MediaHeartbeat.Event.AdComplete event when the ad reaches the end,
// i.e., when the ad completes and finishes playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete, null, null);

// 10. Track the MediaHeartbeat.Event.AdBreakComplete event when all the ads in the
// mid-roll pod finish playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete, null, null);

// Post-roll
MediaObject adBreakInfo =
    MediaHeartbeat.createAdBreakObject(POSTROLL_BREAK_NAME,
        POSTROLL_BREAK_POSITION,
        POSTROLL_BREAK_START_TIME);
MediaObject adInfo =
    MediaHeartbeat.createAdObject(POSTROLL_AD_NAME,
        POSTROLL_AD_ID,
        POSTROLL_AD_POSITION,
        POSTROLL_AD_LENGTH);

// Context ad data
HashMap<String, String> adMetadata = new HashMap<String, String>();
adMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
adMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 11. Track the MediaHeartbeat.Event.AdBreakStart event when the post-roll pod
// starts to play.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo, null);
// 12. Track the MediaHeartbeat.Event.AdStart event when the post-roll pod's ad starts to play.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

// 13. Track the MediaHeartbeat.Event.AdComplete event when the ad reaches the end, i.e., when the ad completes and finishes playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete, null, null);

_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete, null, null);

// 15. Call trackComplete() when the playback reaches the end, i.e., when the video completes and finishes playing.
_mediaHeartbeat.trackComplete();

// 16. Call trackSessionEnd() when the playback session is over. This method must be called even if the user does not watch the video to completion.
_mediaHeartbeat.trackSessionEnd();

• iOS

To view this scenario in iOS, set up the following code:

// Set up mediaObject
ADBMediaObject *mediaObject =
[ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
length:MEDIA_LENGTH
streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData =
[[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used, i.e., there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// Pre-roll
ADBMediaObject *adBreakInfo =
[ADBMediaHeartbeat createAdBreakObjectWithName:AD_BREAK_NAME
position:AD_BREAK_POSITION
startTime:AD_BREAK_START_TIME];

ADBMediaObject *adInfo =
[ADBMediaHeartbeat createAdObjectWithName:AD_NAME
adId:AD_ID
position:AD_POSITION
length:AD_LENGTH];
// Context ad data
NSMutableDictionary *adDictionary =
    [[NSMutableDictionary alloc] init];
[adDictionary setObject:@"custom-val1" forKey:@"custom-key1"];
[adDictionary setObject:@"custom-val2" forKey:@"custom-key2"];  

// 2. Track the ADBMediaHeartbeatEventAdBreakStart event when the
//    pre-roll pod starts to play. Note that since this is a pre-roll,
//    you must track the ADBMediaHeartbeatEventAdBreakStart event
//    before you call trackPlay.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakStart
    mediaObject:adBreakObject
data:adDictionary];

// 3. Track the ADBMediaHeartbeatEventAdStart when the pre-roll
//    pod’s ad starts to play. Note that since this is a pre-roll,
//    you must track the ADBMediaHeartbeatEventAdStart before you
//    call trackPlay.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdStart
    mediaObject:adObject
data:adDictionary];

// 4. Call trackPlay when the playback actually starts, i.e., when
//    the first frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];

// 5. Track the ADBMediaHeartbeatEventAdComplete event when the ad
//    reaches the end, i.e., when it completes and finishes playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdComplete
    mediaObject:nil
data:nil];

// 6. Track the ADBMediaHeartbeatEventAdBreakComplete event when all
//    of the ads in the pod finish playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakComplete
    mediaObject:nil
data:nil];

// Mid-roll
ADBMediaObject *adBreakInfo =
    [ADBMediaHeartbeat createAdBreakObjectWithName:MIDROLL_BREAK_NAME
        position:MIDROLL_BREAK_POSITION
        startTime:MIDROLL_BREAK_START_TIME];
ADBMediaObject *adInfo =
    [ADBMediaHeartbeat createAdObjectWithName:MIDROLL_AD_NAME
        adId:MIDROLL_AD_ID position:MIDROLL_AD_POSITION
        length:MIDROLL_AD_LENGTH];

// context ad data
NSMutableDictionary *midrollAdDictionary =
    [[NSMutableDictionary alloc] init];
[adDictionary setObject:@"custom-val1" forKey:@"custom-key1"];
[adDictionary setObject:@"custom-val2" forKey:@"custom-key2"];  

// 7. Track the ADBMediaHeartbeatEventAdBreakStart event when the mid-roll pod
//    starts to play.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakStart
    mediaObject:adBreakObject
data:nil];
8. Track the ADBMediaHeartbeatEventAdStart event when the mid-roll pod's ad starts to play.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdStart mediaObject:adObject data:midrollAdDictionary];

9. Track the ADBMediaHeartbeatEventAdComplete event when the ad reaches the end, i.e., when it completes and finishes playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdComplete mediaObject:nil data:nil];

10. Track the ADBMediaHeartbeatEventAdBreakComplete event when all the ads in the mid-roll pod finish playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakComplete mediaObject:nil data:nil];

---

11. Track the ADBMediaHeartbeatEventAdBreakStart event when the post-roll pod starts to play.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakStart mediaObject:adBreakObject data:nil];

12. Track the ADBMediaHeartbeatEventAdStart event when the post-roll pod's ad starts to play.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdStart mediaObject:adObject data:postrollAdDictionary];

13. Track the ADBMediaHeartbeatEventAdComplete event when the post-roll pod's ad finishes playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdComplete mediaObject:nil data:nil];

14. Track the ADBMediaHeartbeatEventAdBreakComplete event when all the ads in the post-roll pod finish playing.
// 15. Call trackComplete when the playback reaches the end, // i.e., when the video completes and finishes playing. 
[mediaHeartbeat trackComplete];

// 16. Call trackSessionEnd when the playback session is over. This method // must be called even if the user does not watch the video to completion. 
[mediaHeartbeat trackSessionEnd];

• JavaScript -
To view this scenario in JavaScript, enter the following text:

```javascript
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

var videoMetadata = {  
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay is used, // i.e., when there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// Preroll
var adBreakInfo =  
    MediaHeartbeat.createAdBreakObject(ADBREAK_NAME,
    ADBREAK_POSITION,
    ADBREAK_START_TIME);

var adInfo =  
    MediaHeartbeat.createAdObject(AD_NAME,
    AD_ID,
    AD_POSITION,
    AD_LENGTH);

// Custom ad metadata
var adMetadata = {  
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2
};

// 2. Track the MediaHeartbeat.Event.AdBreakStart event when the preroll pod // starts to play. Note that since this is a preroll, you must track the // MediaHeartbeat.Event.AdBreakStart event before you call trackPlay().
this._trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo, null);

// 3. Track the MediaHeartbeat.Event.AdStart event when the preroll pod's ad
// starts to play. Note that since this is a preroll, you must track the
// MediaHeartbeat.Event.AdStart event before you call trackPlay().
this._heartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

.......

// 4. Call trackPlay() when the main content actually starts, i.e., when the
// first frame of the video content is rendered on the screen.
this._mediaHeartbeat.trackPlay();

.......

// 5. Track the MediaHeartbeat.Event.AdComplete event when the ad reaches the end,
// i.e., when the ad completes and finishes playing.
this._heartbeat.trackEvent(MediaHeartbeat.Event.AdComplete, null, null);

.......

// 6. Track the MediaHeartbeat.Event.AdBreakComplete event when all of the ads in
// the pod finish playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete);

.......

// Midroll
var adBreakInfo =
    MediaHeartbeat.createAdBreakObject(MIDROLL_BREAK_NAME,
                                        MIDROLL_BREAK_POSITION,
                                        MIDROLL_BREAK_START_TIME);

var adInfo =
    MediaHeartbeat.createAdObject(MIDROLL_AD_NAME,
                                   MIDROLL_AD_ID,
                                   MIDROLL_AD_POSITION,
                                   MIDROLL_AD_LENGTH);

// Custom ad metadata
var adMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2
};

// 7. Track the MediaHeartbeat.Event.AdBreakStart event when the
// midroll pod starts to play.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo);

.......

// 8. Track the MediaHeartbeat.Event.AdStart event when the midroll
// pod's ad starts to play.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart,
                                adInfo,
                                adMetadata);

.......

// 9. Track the MediaHeartbeat.Event.AdComplete event when the ad
// reaches the end, i.e., when the ad completes and finishes playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete);

.......
// 10. Track the MediaHeartbeat.Event.AdBreakComplete event when all of
//     the ads in the midroll pod finish playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete);

// Set up mediaObject
var mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

var videoMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay
//    is used, i.e., when there’s an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// Preroll
var adBreakInfo =
    MediaHeartbeat.createAdBreakObject(ADBREAK_NAME,
                                        ADBREAK_POSITION,
                                        ADBREAK_START_TIME);

var adInfo =
    MediaHeartbeat.createAdObject(AD_NAME,
                                   AD_ID,
                                   AD_POSITION,
                                   AD_LENGTH);

// Custom ad metadata
var adMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2
};

// 2. Track the MediaHeartbeat.Event.AdBreakStart event when the preroll pod
//    starts to play. Note that since this is a preroll, you must track the
//    MediaHeartbeat.Event.AdBreakStart event before you call trackPlay().
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo);

// 3. Track the MediaHeartbeat.Event.AdStart event when the preroll pod’s
//    ad starts to play. Note that since this is a preroll, you must track
//    the MediaHeartbeat.Event.AdStart event before you call trackPlay().
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

// 4. Call trackPlay() when the playback actually starts, i.e., when the first
//    frame of the main content is rendered on the screen.
_mediaHeartbeat.trackPlay();
// 5. Track the MediaHeartbeat.Event.AdComplete event when the ad reaches
//    the end, i.e., when the ad completes and finishes playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete);

// 6. Track the MediaHeartbeat.Event.AdBreakComplete event when all
//    of the ads in the pod finish playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete);

// Mid-roll
var adBreakInfo = 
    MediaHeartbeat.createAdBreakObject(MIDROLL_BREAK_NAME,
        MIDROLL_BREAK_POSITION,
        MIDROLL_BREAK_START_TIME);
var adInfo = 
    MediaHeartbeat.createAdObject(MIDROLL_AD_NAME,
        MIDROLL_AD_ID,
        MIDROLL_AD_POSITION,
        MIDROLL_AD_LENGTH);

// Custom ad metadata
var adMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2
};

// 7. Track the MediaHeartbeat.Event.AdBreakStart event when the midroll
//    pod starts to play.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo);

// 8. Track the MediaHeartbeat.Event.AdStart event when the midroll pod's
//    ad starts to play.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

// 9. Track the MediaHeartbeat.Event.AdComplete event when the ad reaches
//    the end, i.e., when the ad completes and finishes playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete);

// 10. Track the MediaHeartbeat.Event.AdBreakComplete event when all
//     of the ads in the midroll pod finish playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete);
VOD playback with skipped ads

Scenario
This scenario comprises VOD content playback with a skipped ad.

Table 13: One VOD with a skipped pre-roll ad
This is the same scenario as VOD playback with pre-roll ads, except the application has a provision to let the user skip the ad, on the click of a skip button perhaps.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks Play</td>
<td>trackSessionStart()</td>
<td>Analytics Content Start, Heartbeat Content Start</td>
<td>The measurement library is unaware that</td>
</tr>
<tr>
<td>Trigger</td>
<td>Heartbeat method</td>
<td>Network calls</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>---------------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| The ad starts. | • `trackEvent:AdBreakStart`  
• `trackEvent:AdStart` | Analytics Ad Start, Heartbeat Ad Start | there is a pre-roll ad. These network calls are still exactly the same as VOD playback with no ads scenario. |
| The first frame of the ad is played. | `trackPlay()` | Heartbeat Ad Play | When ad content plays before main content, the heartbeats will start when the ad starts to play. |
| The ad plays. | | Ad Heartbeats | |
| The ad is skipped. | `trackEvent:trackAdSkip` | | There is no ad complete network call. |
| The content plays. | | Content Heartbeats | These network calls are exactly the same as the VOD playback with no ads scenario. |
| The content completes playing. | `trackComplete()` | Heartbeat Content Complete | This network call is exactly the same as the VOD playback with no ads scenario. |
| The session is over. | `trackSessionEnd()` | | SessionEnd means the end of a viewing session. This API must be called even if the user does not watch the video to completion. |

**Parameters**

The parameters are identical to the parameters in the VOD playback with pre-roll ads scenario, except there is no ad complete and no ad-break complete call.
Sample Code

**Android**

To view this scenario in Android, set up the following code:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
//    i.e., there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......

// Pre-roll
MediaObject adBreakInfo =
    MediaHeartbeat.createAdBreakObject(ADBREAK_NAME,
    ADBREAK_POSITION,
    ADBREAK_START_TIME);

MediaObject adInfo =
    MediaHeartbeat.createAdObject(AD_NAME,
    AD_ID,
    AD_POSITION,
    AD_LENGTH);

// Context ad data
HashMap<String, String> adMetadata = new HashMap<String, String>();
adMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
adMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 2. Track the MediaHeartbeat.Event.AdBreakStart event when the pre-roll pod starts to play.
// Note that since this is a pre-roll, track the "MediaHeartbeat.Event.AdBreakStart"
// event before you call trackPlay().
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo, null);

......

// 3. Track the MediaHeartbeat.Event.AdStart event when the pre-roll pod's ad starts to play.
// Note that since this is a pre-roll, track the "MediaHeartbeat.Event.AdStart" event
// before you call trackPlay().
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);
```
When the user clicks Play:
ADBMediaObject *mediaObject =
   [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
    length:MEDIA_LENGTH
    streamType:ADBMediaHeartbeatStreamTypeVOD];
NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
// i.e., there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// Pre-roll
ADBMediaObject *adBreakInfo =
   [ADBMediaHeartbeat createAdBreakObjectWithName:AD_BREAK_NAME
    position:AD_BREAK_POSITION
    startTime:AD_BREAK_START_TIME];
ADBMediaObject *adInfo =
   [ADBMediaHeartbeat createAdObjectWithName:AD_NAME
    adId:AD_ID
    position:AD_POSITION
    length:AD_LENGTH];

// Context ad data
NSMutableDictionary *adDictionary = [[NSMutableDictionary alloc] init];
[adDictionary setObject:@"custom-val1" forKey:@"custom-key1"];
[adDictionary setObject:@"custom-val2" forKey:@"custom-key2"];
// 2. Track the ADBMediaHeartbeatEventAdBreakStart event when the pre-roll pod
// starts to play. Note that since this is a pre-roll, you must track the
// ADBMediaHeartbeatEventAdBreakStart event before you call trackPlay.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdBreakStart
 mediaObject:adBreakObject
data:nil];
........
........

// 3. Track the ADBMediaHeartbeatEventAdStart event when the pre-roll pod's ad
// starts to play. Note that since this is a pre-roll, track the
// ADBMediaHeartbeatEventAdStart event before you call trackPlay.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdStart
 mediaObject:adObject
data:adDictionary];
........
........

// 4. Call trackPlay when the playback actually starts, i.e., when the first
// frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];
........
........

// 5. Track the ADBMediaHeartbeatEventAdSkip event when the user intends to
// and is able to skip an ad. For example, this could be tied to a
// "skip ad" button onClick handler. The application could have the viewer
// land in main content post ad.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventAdSkip mediaObject:nil data:nil];
........
........

// 6. Call trackComplete when the playback reaches the end, i.e., when the video
// completes and finishes playing.
[_mediaHeartbeat trackComplete];
........
........

// 7. Call trackSessionEnd when the playback session is over. This method must
// be called even if the user does not watch the video to completion.
[_mediaHeartbeat trackSessionEnd];
........
........

• JavaScript -

To view this scenario in JavaScript, enter the following text:

// Set up mediaObject
var mediaInfo =
    MediaHeartbeat.createMediaObject(Configuration.MEDIA_NAME,
        Configuration.MEDIA_ID,
        Configuration.MEDIA_LENGTH,
        MediaHeartbeat.StreamType.VOD);

var videoMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay is used,
// i.e., there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);
........
........

// Preroll
var adBreakInfo = MediaHeartbeat.createAdBreakObject(ADBREAK_NAME, ADBREAK_POSITION, ADBREAK_START_TIME);

MediaObject adInfo = MediaHeartbeat.createAdObject(AD_NAME, AD_ID, AD_POSITION, AD_LENGTH);

// context ad data
var adMetadata = {
    CUSTOM_KEY_1: CUSTOM_VAL_1,
    CUSTOM_KEY_2: CUSTOM_VAL_2
};

// 2. Track the MediaHeartbeat.Event.AdBreakStart event when the preroll pod starts to play.
//    Since this is a preroll, you must track the MediaHeartbeat.Event.AdBreakStart event
//    before calling trackPlay().
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo);

// 3. Track the MediaHeartbeat.Event.AdStart event when the preroll pod's ad starts to play.
//    Since this is a preroll, you must track the MediaHeartbeat.Event.AdStart event before
//    calling trackPlay().
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adMetadata);

// 4. Call trackPlay() when the playback actually starts, i.e., when the first frame of
//    the main content is rendered on the screen.
this._mediaHeartbeat.trackPlay();

// 5. Track the MediaHeartbeat.Event.AdSkip event when the user intends to (and can)
//    skip the ad. For example, this could be tied to a "skip ad" button onClick handler.
//    The application could have the viewer land in the main content post ad.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdSkip);

// 6. Call trackComplete() when the playback reaches the end, i.e., playback completes
//    and finishes playing.
this._mediaHeartbeat.trackComplete();

// 7. Call trackSessionEnd() when the playback session is over. This method must be called even
//    if the user does not watch the video to completion.
this._mediaHeartbeat.trackSessionEnd();
**VOD playback with one chapter**

**Scenario**

In this scenario, a portion of the VOD content is marked as a chapter.

Unless specified, the network calls in this scenario are the same as the calls in the *VOD playback with no ads* scenario. The network call happens at the same time, but the payload is different.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks <strong>Play</strong></td>
<td><strong>trackSessionStart</strong></td>
<td>Analytics Content Start, Heartbeat Content Start</td>
<td>We have not yet told the measurement library that there is a pre-roll ad, so these network calls are still exactly the same as Single VoD.</td>
</tr>
<tr>
<td>The chapter starts.</td>
<td><strong>trackEvent:ChapterStart</strong></td>
<td>Heartbeat Chapter Start</td>
<td></td>
</tr>
<tr>
<td>First frame of the chapter plays.</td>
<td><strong>trackPlay</strong></td>
<td>Heartbeat Content Play</td>
<td>When chapter content plays before main content, the Heartbeats start when the chapter starts.</td>
</tr>
<tr>
<td>The chapter plays.</td>
<td></td>
<td>Chapter Heartbeats</td>
<td></td>
</tr>
<tr>
<td>The chapter is complete.</td>
<td><strong>trackEvent:trackChapterComplete</strong></td>
<td>Heartbeat Chapter Complete</td>
<td>This is when the end of the chapter is reached.</td>
</tr>
<tr>
<td>The content plays.</td>
<td></td>
<td>Content Heartbeats</td>
<td>This network call is exactly the same as the <em>VOD playback with no ads</em> scenario.</td>
</tr>
<tr>
<td>The content is complete.</td>
<td><strong>trackComplete</strong></td>
<td>Heartbeat Content Complete</td>
<td>This network call is exactly the same as the <em>VOD playback with no ads</em> scenario.</td>
</tr>
<tr>
<td>The session is over.</td>
<td><strong>trackSessionEnd</strong></td>
<td></td>
<td>SessionEnd means that the end of a viewing session has been reached. This API must be called even if the user does not watch the video to completion.</td>
</tr>
</tbody>
</table>

**Parameters**

When chapter playback begins, a **Heartbeat Chapter Start** call is sent. If the beginning of the chapter does not coincide with the 10-second timer, the **Heartbeat Chapter Start** call is delayed by a few seconds, and the call goes to the next 10-second interval.
When this happens, a Content Heartbeat call goes out in the same interval. You can differentiate between the two by examining the event type and the asset type:

**Table 14: Heartbeat Chapter Start**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;chapter_start&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;main&quot;</td>
<td></td>
</tr>
<tr>
<td>s:stream:chapter_*</td>
<td></td>
<td>Stream information that is specific to the chapter data.</td>
</tr>
<tr>
<td>s:meta:*</td>
<td></td>
<td>Chapter with specific context data.</td>
</tr>
</tbody>
</table>

**Sample code, chapter in the middle**

In this scenario, part of the VOD content is a chapter.

```
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
// i.e., there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......

// 2. Call trackPlay() when the playback actually starts, i.e., first frame of the
// ad video is rendered on the screen.
_mediaHeartbeat.trackPlay();

......
```

- **Android** -

To view this scenario in Android, set up the following code:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
// i.e., there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......

// 2. Call trackPlay() when the playback actually starts, i.e., first frame of the
// ad video is rendered on the screen.
_mediaHeartbeat.trackPlay();

......
```
// Chapter
HashMap<String, String> chapterMetadata = new HashMap<String, String>();
chapterMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);

MediaObject chapterDataInfo =
  MediaHeartbeat.createChapterObject(CHAPTER_NAME,
  CHAPTER_POSITION,
  CHAPTER_LENGTH,
  CHAPTER_START_TIME);

// 3. Track the MediaHeartbeat.Event.ChapterStart event when the chapter starts to play.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart, chapterDataInfo, chapterMetadata);

// 4. Track the MediaHeartbeat.Event.ChapterComplete event when the chapter finishes playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterComplete, null, null);

// 5. Call trackComplete() when the playback reaches the end, i.e., completes and finishes playing.
_mediaHeartbeat.trackComplete();

// 6. Call trackSessionEnd() when the playback session is over. This method must be called even
//    if the user does not watch the video to completion.
_mediaHeartbeat.trackSessionEnd();

• iOS -

To view this scenario in iOS, set up the following code:

when the user clicks Play
ADBMediaObject *mediaObject =
  [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
   length:MEDIA_LENGTH
   streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
//    i.e., when there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// 2. Call trackPlay when the playback actually starts, i.e., when the
//    first frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];

// Chapter
NSMutableDictionary *chapterContextData = [[NSMutableDictionary alloc] init];
[chapterContextData setObject:CONTEXT_DATA_VALUE forKey:CONTEXT_DATA_KEY];

id chapterInfo =
  [ADBMediaHeartbeat createChapterObjectWithName:CHAPTER_NAME
   position:CHAPTER_POSITION]
// 3. Track the ADBMediaHeartbeatEventChapterStart event when the chapter
// starts to play.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventChapterStart
mediaObject:chapterInfo
data:chapterContextData];

// 4. Track the ADBMediaHeartbeatEventChapterComplete event when the chapter
// finishes playing.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventChapterComplete
mediaObject:nil
data:nil];

// 5. Call trackComplete when the playback reaches the end, i.e., when the
// video completes and finishes playing.
[_mediaHeartbeat trackComplete];

// 6. Call trackSessionEnd when the playback session is over. This method must
// be called even if the user does not watch the video to completion.
[_mediaHeartbeat trackSessionEnd];

• JavaScript -

To view this scenario in JavaScript, enter the following text:

// Set up mediaObject
var mediaInfo = MediaHeartbeat.createMediaObject(
  Configuration.MEDIA_NAME,
  Configuration.MEDIA_ID,
  Configuration.MEDIA_LENGTH,
  MediaHeartbeat.StreamType.VOD
);

var videoMetadata = {
  CUSTOM_KEY_1 : CUSTOM_VAL_1,
  CUSTOM_KEY_2 : CUSTOM_VAL_2,
  CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart when Play is clicked or if autoplay is used,
// i.e., when there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// Chapter
var chapterMetadata = {
  CUSTOM_KEY_1 : CUSTOM_VAL_1
};

var chapterDataInfo =
  MediaHeartbeat.createChapterObject(CHAPTER_NAME,
  CHAPTER_POSITION,
  CHAPTER_LENGTH,
  CHAPTER_START_TIME);

// 2. Track the MediaHeartbeat.Event.ChapterStart event when the chapter
// starts to play.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart,
// 3. Call trackPlay() when the playback actually starts, i.e., when the first frame of the main content is rendered on the screen.
this._mediaHeartbeat.trackPlay();

// 4. Track the MediaHeartbeat.Event.ChapterComplete event when the chapter finishes playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterComplete);

// 5. Call trackComplete() when the playback reaches the end, i.e., when playback completes and finishes playing.
this._mediaHeartbeat.trackComplete();

// 6. Call trackSessionEnd() when the playback session is over. This method must be called even if the user does not watch the video to completion.
this._mediaHeartbeat.trackSessionEnd();

Sample code, chapter at the beginning

In this scenario, VOD content is played back with one chapter at the beginning of the playback.

• Android -

To view this scenario in Android, set up the following code:

// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
//    i.e., there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......
......

// 2. Call trackPlay() when the playback actually starts, i.e., first frame of the
//    main content is rendered on the screen.
_mediaHeartbeat.trackPlay();

......
......

// Chapter
HashMap<String, String> chapterMetadata = new HashMap<String, String>();
chapterMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
MediaObject chapterDataInfo =
    MediaHeartbeat.createChapterObject(CHAPTER_NAME,
    CHAPTER_POSITION,
    CHAPTER_LENGTH,
    CHAPTER_START_TIME);

// 3. Track the MediaHeartbeat.Event.ChapterStart event when the chapter starts to play.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart, chapterDataInfo, chapterMetadata);

......
......

// 4. Track the MediaHeartbeat.Event.ChapterComplete event when the chapter finishes playing.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterComplete, null, null);

......
......

// 5. Call trackComplete() when the playback reaches the end, i.e., when the video completes
//    and finishes playing.
_mediaHeartbeat.trackComplete();

......
......

// 6. Call trackSessionEnd() when the playback session is over. This method must be called
//    even if the user does not watch the video to completion.
_mediaHeartbeat.trackSessionEnd();

......
......

• iOS -

To view this scenario in iOS, set up the following code:

when the user clicks Play
ADBMediaObject *mediaObject =
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
        length:MEDIA_LENGTH
        streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
// i.e., there is an intent to start playback.
[mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// Chapter
NSMutableDictionary *chapterContextData = [[NSMutableDictionary alloc] init];
[chapterContextData setObject:CONTEXT_DATA_VALUE forKey:CONTEXT_DATA_KEY];

id chapterInfo =
 [ADBMediaHeartbeat createChapterObjectWithName:CHAPTER_NAME
 position:CHAPTER_POSITION
 length:CHAPTER_LENGTH
 startTime:CHAPTER_START_TIME];

// 2. Call ADBMediaHeartbeatEventChapterStart when the chapter starts.
[mediaHeartbeat trackEvent:ADBMediaHeartbeatEventChapterStart
 mediaObject:chapterInfo
 data:chapterContextData];

// 3. Call trackPlay when the playback actually starts, i.e., when the
// first frame of the main content is rendered on the screen.
[mediaHeartbeat trackPlay];

// 4. Call ADBMediaHeartbeatEventChapterComplete when the chapter starts.
[mediaHeartbeat trackEvent:ADBMediaHeartbeatEventChapterComplete
 mediaObject:nil
 data:nil];

// 5. Call trackComplete when the playback reaches the end, i.e., when the
// video completes and finishes playing.
[mediaHeartbeat trackComplete];

// 6. Call trackSessionEnd when the playback session is over. This method
// must be called even if the user does not watch the video to completion.
[mediaHeartbeat trackSessionEnd];

• JavaScript -

To view this scenario in JavaScript, enter the following text:

// Set up mediaObject
var mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

var videoMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay is used,
// i.e., when there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);
.....
.....

// Chapter
var chapterMetadata = {
  CUSTOM_KEY_1 : CUSTOM_VAL_1
};

var chapterDataInfo =
  MediaHeartbeat.createChapterObject(CHAPTER_NAME,
  CHAPTER_POSITION,
  CHAPTER_LENGTH,
  CHAPTER_START_TIME);

// 2. Track the MediaHeartbeat.Event.ChapterStart event when the chapter starts to play.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart,
  chapterDataInfo,
  chapterMetadata);

.....
.....

// 3. Call trackPlay() when the playback actually starts, i.e., when the first
// frame of the main content is rendered on the screen.
this._mediaHeartbeat.trackPlay();

.....
.....

// 4. Track the MediaHeartbeat.Event.ChapterComplete event when the chapter
// finishes playing.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterComplete);

.....
.....

// 5. Call trackComplete() when the playback reaches the end, i.e., when playback
// completes and finishes playing.
this._mediaHeartbeat.trackComplete();

.....
.....

// 6. Call trackSessionEnd() when the playback session is over. This method must be
// called even if the user does not watch the video to completion.
this._mediaHeartbeat.trackSessionEnd();

.....
.....

VOD playback with a skipped chapter

Scenario

In this scenario, the user skips a chapter in the main content.

This is the same scenario as VOD playback with one chapter, except the user in this case intends to seek out of the chapter thereby skipping it to land into main content.
<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks Play</td>
<td>trackSessionStart</td>
<td>Analytics Content Start, Heartbeat Content Start</td>
<td>The measurement library is unaware that there is a pre-roll ad. These network calls are still exactly the same as scenario.</td>
</tr>
<tr>
<td>The chapter starts.</td>
<td>trackEvent:ChapterStart</td>
<td>Heartbeat Chapter Start</td>
<td></td>
</tr>
<tr>
<td>The first frame of the chapter is played.</td>
<td>trackPlay</td>
<td>Heartbeat Chapter Play</td>
<td>When chapter content plays before main content, we want to start the heartbeats when the chapter starts.</td>
</tr>
<tr>
<td>The chapter plays.</td>
<td></td>
<td>Chapter Heartbeats</td>
<td></td>
</tr>
<tr>
<td>Seek begins to skip the first chapter.</td>
<td>trackEvent:trackSeekStart</td>
<td></td>
<td>No heartbeats during seeking</td>
</tr>
<tr>
<td>The seek is complete.</td>
<td>trackEvent:trackSeekComplete</td>
<td></td>
<td>Heartbeats would resume post this.</td>
</tr>
<tr>
<td>The application realizes that the user has seeked out of the regular chapter boundary.</td>
<td>trackEvent:trackChapterSkip</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The content plays.</td>
<td></td>
<td>Content Heartbeats</td>
<td></td>
</tr>
<tr>
<td>The content completes playing.</td>
<td>trackComplete</td>
<td>Heartbeat Content Complete</td>
<td>This network call is exactly the same as the scenario.</td>
</tr>
<tr>
<td>The session is over.</td>
<td>trackSessionEnd</td>
<td></td>
<td>SessionEnd means the end of a viewing session. This API must be called even if the user does not watch the video to completion.</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters used during chapter playback are identical to the parameters in the VOD playback with one chapter scenario, except that there is no chapter complete network call.

**Sample code**

```
1. trackSessionStart
2. trackEvent: ADBMediaHeartbeatEventChapterStart
3. trackPlay
4. trackEvent: ADBMediaHeartbeatEventSeekStart
5. trackEvent: ADBMediaHeartbeatEventSeekComplete
6. trackEvent: ADBMediaHeartbeatEventChapterSkip
7. trackEvent: ADBMediaHeartbeatEventChapterComplete
8. trackSessionEnd
```

- Android -
To view this scenario in Android, set up the following code:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
// i.e., there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......
......

// Chapter
HashMap<String, String> chapterMetadata = new HashMap<String, String>();
chapterMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
MediaObject chapterDataInfo =
    MediaHeartbeat.createChapterObject(CHAPTER_NAME,
    CHAPTER_POSITION,
    CHAPTER_LENGTH,
    CHAPTER_START_TIME);

// 2. Track the MediaHeartbeat.Event.ChapterStart event when the chapter starts to play.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart,
    chapterDataInfo,
    chapterMetadata);

......
......

// 3. Call trackPlay() when the playback actually starts, i.e., when the first frame
// of the main content is rendered on the screen.
_mediaHeartbeat.trackPlay();

......
......

// 4. Track the MediaHeartbeat.Event.SeekStart event when the user begins to seek out
// of the chapter with the intent to skip it.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekStart, null, null);

......
......

// 5. Track the MediaHeartbeat.Event.SeekComplete event when the user seeks out of the
// chapter with the intent to skip it.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete, null, null);

......
......

// 6. Track the MediaHeartbeat.Event.ChapterSkip event because the user skipped the
// chapter by seeking out of it in the steps above.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterSkip, null, null);

......
......

// 7. Call trackComplete() when the playback reaches the end, i.e., when the video
// completes and finishes playing.
```
mediaHeartbeat.trackComplete();

........
........

// 8. Call trackSessionEnd() when the playback session is over. This method must be
called even if the user does not watch the video to completion.
mediaHeartbeat.trackSessionEnd();

........
........

* iOS *

To view this scenario in iOS, set up the following code:

// Set up mediaObject
ADBMediaObject *mediaObject =
   [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
    length:MEDIA_LENGTH
    streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
i.e., there's an intent to start playback.
[mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

........
........

// Chapter
NSMutableDictionary *chapterContextData = [[NSMutableDictionary alloc] init];
[chapterContextData setObject:CONTEXT_DATA_VALUE forKey:CONTEXT_DATA_KEY];

id chapterInfo =
   [ADBMediaHeartbeat createChapterObjectWithName:CHAPTER_NAME
    position:CHAPTER_POSITION
    length:CHAPTER_LENGTH
    startTime:CHAPTER_START_TIME];

// 2. Track the ADBMediaHeartbeatEventChapterStart event when the chapter starts.
[mediaHeartbeat trackEvent:ADBMediaHeartbeatEventChapterStart
     mediaObject:chapterInfo
data:chapterContextData];

........
........

// 3. Call trackPlay when the playback actually starts, i.e., when the first
frame of the main content is rendered on the screen.
[mediaHeartbeat trackPlay];

........
........

// 4. Track the trackEvent:ADBMediaHeartbeatEventSeekStart event when the user
begins to seek out of the chapter with the intent to skip it.
[mediaHeartbeat trackEvent:ADBMediaHeartbeatEventSeekStart mediaObject:nil data:nil];

........
........

// 5. Track the trackEvent:ADBMediaHeartbeatEventSeekComplete event when the
user seeks out of the chapter with the intent to skip it.
[mediaHeartbeat trackEvent:ADBMediaHeartbeatEventSeekComplete mediaObject:nil data:nil];

........

// 6. Track the trackEvent:ADBMediaHeartbeatEventChapterSkip event because the
user skipped the chapter by seeking out of it in the steps above.
_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventChapterSkip
    mediaObject:chapterInfo
    data:chapterContextData;

7. Call `trackComplete` when the playback reaches the end, i.e., when the video
    completes and finishes playing.
_mediaHeartbeat trackComplete;

8. Call `trackSessionEnd` when the playback session is over. This method must
    be called even if the user does not watch the video to completion.
_mediaHeartbeat trackSessionEnd;

• JavaScript -

To view this scenario in JavaScript, enter the following text:

```javascript
var mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD);

var videoMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call `trackSessionStart()` when Play is clicked or if autoplay is used, 
// i.e., there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// Chapter
var chapterMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1
};

var chapterDataInfo = 
    MediaHeartbeat.createChapterObject(CHAPTER_NAME,
        CHAPTER_POSITION,
        CHAPTER_LENGTH,
        CHAPTER_START_TIME);

// 2. Track the `MediaHeartbeat.Event.ChapterStart` event when the chapter starts to play.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart,
    chapterDataInfo,
    chapterMetadata);

// 3. Call `trackPlay()` when the playback actually starts, i.e., when the 
// first frame of the main content is rendered on the screen.
this._mediaHeartbeat.trackPlay();
```
// 4. Track the MediaHeartbeat.Event.SeekStart event when the user begins
// to seek out of the chapter with the intent to skip it.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekStart);

// 5. Track the MediaHeartbeat.Event.SeekComplete event when the user seeks
// out of the chapter with the intent to skip it.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete);

// 6. Track the MediaHeartbeat.Event.ChapterSkip event because the user
// skipped the chapter by seeking out of it in the steps above.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterSkip);

// 7. Call trackComplete() when the playback reaches the end, i.e., completes
// and finishes playing.
this._mediaHeartbeat.trackComplete();

// 8. Call trackSessionEnd() when the playback session is over. This method must be
// called even if the user does not watch the video to completion.
this._mediaHeartbeat.trackSessionEnd();

VOD playback with seeking in the main content

Scenario

This scenario comprises seeking in the main content during playback.

This is the same scenario as the VOD playback with no ads scenario, but a part of the content is scrubbed through and a seek is completed from one point in main content to another point.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks Play</td>
<td>trackSessionStart</td>
<td>Analytics Content Start, HeartbeatContent Start</td>
<td>The measurement library is unaware that there is a pre-roll ad, so these network calls are identical to the VOD playback with no ads scenario.</td>
</tr>
<tr>
<td>First frame of the content plays.</td>
<td>trackPlay</td>
<td>Heartbeat Content Play</td>
<td>When chapter content plays before main content, the Heartbeats start when the chapter starts.</td>
</tr>
<tr>
<td>Content plays</td>
<td></td>
<td>Content Heartbeats</td>
<td>This network call is exactly the same as the VOD playback with no ads scenario.</td>
</tr>
<tr>
<td>Trigger</td>
<td>Heartbeat method</td>
<td>Network calls</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>---------------</td>
<td>-------</td>
</tr>
<tr>
<td>User begins seek operation on content</td>
<td>trackSeekStart</td>
<td></td>
<td>No heartbeats go out till seek is complete, for example, trackSeekComplete is called.</td>
</tr>
<tr>
<td>Seek operation completes</td>
<td>trackSeekComplete</td>
<td></td>
<td>Heartbeats begin to go out since seek is complete.</td>
</tr>
<tr>
<td>Content is complete</td>
<td>trackComplete</td>
<td>Heartbeat Content Complete</td>
<td>This network call is exactly the same as the VOD playback with no ads scenario.</td>
</tr>
<tr>
<td>Session Over</td>
<td>trackSessionEnd</td>
<td></td>
<td>SessionEnd means that the end of a viewing session has been reached. This API must be called even if the user does not watch the video to completion.</td>
</tr>
</tbody>
</table>

**Sample Code**

In this scenario, the user is seeking when the main content is being played.

**Android -**

To view this scenario in Android, set up the following code:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used, i.e., there is an intent to start playback.
```
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......
......

// 2. Call trackPlay() when the playback actually starts, i.e., when the first frame of the main content is rendered on the screen.
_mediaHeartbeat.trackPlay();

......
......

// 3. Track the MediaHeartbeat.Event.SeekStart event when the user begins to seek.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekStart, null, null);

......
......

// 4. Track the MediaHeartbeat.Event.SeekComplete event when the user completes seeking
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete, null, null);

......
......

// 5. Call trackComplete() when the playback reaches the end, i.e., when the video completes and finishes playing.
_mediaHeartbeat.trackComplete();

......
......

// 6. Call trackSessionEnd() when the playback session is over. This method must be called even if the user does not watch the video to completion.
_mediaHeartbeat.trackSessionEnd();

......
......

• iOS -

To view this scenario in iOS, set up the following code:

// Set up mediaObject
ADBMediaObject *mediaObject =
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME o
     length:MEDIA_LENGTH
     streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used, i.e., there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

......
......

// 2. Call trackPlay when the playback actually starts, i.e., when the first frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];

......
......

// 3. Track the trackEvent:ADBMediaHeartbeatEventSeekStart event when the user begins to seek out of the chapter with the intent to skip it.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventSeekStart
     mediaObject:nil
     data:nil];
// 4. Track the trackEvent:ADBMediaHeartbeatEventSeekComplete event when the user seeks out of the chapter with the intent to skip it.
[mediaHeartbeat trackEvent:ADBMediaHeartbeatEventSeekComplete
mediaObject:nil
data:nil];

// 5. Call trackComplete when the playback reaches the end, i.e., completes and finishes playing.
[mediaHeartbeat trackComplete];

// 6. Call trackSessionEnd when the playback session is over. This method must be called even if the user does not watch the video to completion.
[mediaHeartbeat trackSessionEnd];

• JavaScript -

To view this scenario, enter the following text:

// Set up mediaObject
var mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

var videoMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay is used, i.e., there’s an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// 2. Call trackPlay() when the playback actually starts, i.e., when the first frame of the ad video is rendered on the screen.
this._mediaHeartbeat.trackPlay();

// 3. Track the MediaHeartbeat.Event.SeekStart event when the user begins to seek.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekStart);

// 4. Track the MediaHeartbeat.Event.SeekComplete event when the user completes seeking.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete);
/// 5. Call trackComplete() when the playback reaches the end, i.e., when
/// playback completes and finishes playing.
this._mediaHeartbeat.trackComplete();


/// 6. Call trackSessionEnd() when the playback session is over. This method must be called
/// even if the user does not watch the video to completion.
this._mediaHeartbeat.trackSessionEnd();


VOD playback with buffering

Scenario
In this scenario, some buffering occurs when VOD content is played back.

Unless specified, the network calls in this scenario are the same as the calls in the VOD playback with no ads scenario.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks <strong>Play</strong></td>
<td>trackSessionStart</td>
<td>Analytics Content Start, Heartbeat Content Start</td>
<td>This can be a user clicking <strong>Play</strong> or an auto-play event.</td>
</tr>
<tr>
<td>The first frame of the video plays.</td>
<td>trackPlay</td>
<td>Heartbeat Content Play</td>
<td>This method triggers the timer. Heartbeats are sent every 10 seconds as long as playback continues.</td>
</tr>
<tr>
<td>The content plays.</td>
<td></td>
<td>Content Heartbeats</td>
<td></td>
</tr>
<tr>
<td>The buffering starts.</td>
<td>trackEvent:BufferStart</td>
<td>Heartbeat Buffer</td>
<td></td>
</tr>
<tr>
<td>The content is buffered.</td>
<td></td>
<td>Content Heartbeats</td>
<td></td>
</tr>
<tr>
<td>The buffering completes.</td>
<td>trackEvent:BufferComplete</td>
<td>Heartbeat Buffer, Heartbeat Play</td>
<td></td>
</tr>
<tr>
<td>The content plays.</td>
<td></td>
<td>Content Heartbeats</td>
<td></td>
</tr>
<tr>
<td>The content completes playing.</td>
<td>trackComplete</td>
<td>Heartbeat Content Complete</td>
<td>The end of the playhead was reached.</td>
</tr>
<tr>
<td>The session is over.</td>
<td>trackSessionEnd</td>
<td></td>
<td>SessionEnd means the end of a viewing session. This API must be called even if the user does not watch the video to completion.</td>
</tr>
</tbody>
</table>
Parameters

Table 15: Heartbeat Buffer

This scenario refers to a rebuffer event, which is when buffering occurs in addition to the initial buffering period that occurred when playback first began. All call query parameters are identical to the play heartbeats, except that the event:type (with a value of buffer after a trackEvent:BufferStart) is called until trackEvent:BufferComplete or a trackPlay method is called.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;buffer&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Sample Code

In this scenario, buffering occurs when the VOD content is played back.

• Android -

To view this scenario in Android, set up the following code:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_KEY_1, CUSTOM_VAL_1);
videoMetadata.put(CUSTOM_KEY_2, CUSTOM_VAL_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
//    i.e., there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......
......

// 2. Call trackPlay() when the playback actually starts, i.e., when the first
//    frame of the main content is rendered on the screen.
_mediaHeartbeat.trackPlay();

......
......
```
// 3. Track the MediaHeartbeat.Event.BufferStart event when the video player
// goes into the buffering state and begins to buffer content.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferStart, null, null);

// 4. Track the MediaHeartbeat.Event.BufferComplete event when the video player
// goes into the buffering state and begins to buffer content.
_mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferComplete, null, null);

// 5. Call trackComplete() when the playback reaches the end, i.e., when the
// video completes and finishes playing.
_mediaHeartbeat.trackComplete();

// 6. Call trackSessionEnd() when the playback session is over. This method must
// be called even if the user does not watch the video to completion.
_mediaHeartbeat.trackSessionEnd();

• iOS -

To view this scenario in iOS, set up the following code:

ADBMediaObject *mediaObject = 
[ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
    length:MEDIA_LENGTH
    streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData = 
[[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
// i.e., there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// 2. Call trackPlay when the playback actually starts, i.e., when the first
// frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];

// 3. Track the trackEvent:ADBMediaHeartbeatEventBufferStart event when the
// video player goes in buffering state and begins to buffer content.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventBufferStart
    mediaObject:nil
    data:nil];

// 4. Track the trackEvent:ADBMediaHeartbeatEventBufferComplete event when
// the video player goes in buffering state and begins to buffer content.
[_mediaHeartbeat trackEvent:ADBMediaHeartbeatEventBufferComplete
    mediaObject:nil
    data:nil];
5. Call trackComplete when the playback reaches the end, i.e., when the video completes and finishes playing.

```
[_mediaHeartbeat trackComplete];
```

........

6. Call trackSessionEnd when the playback session is over. This method must be called even if the user does not watch the video to completion.

```
[_mediaHeartbeat trackSessionEnd];
```

........

**JavaScript**

To view this scenario, enter the following text:

```
// Set up mediaObject

var mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

var videoMetadata = {
    CUSTOM_KEY_1 : CUSTOM_VAL_1,
    CUSTOM_KEY_2 : CUSTOM_VAL_2,
    CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay is used, i.e., there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

........

// 2. Call trackPlay() when the playback actually starts, i.e., when the first frame of the ad video is rendered on the screen.
this._mediaHeartbeat.trackPlay();

........

// 3. Track event MediaHeartbeat.Event.BufferStart when the video player goes into the buffering state and begins to buffer content.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferStart);

........

// 4. Track the MediaHeartbeat.Event.BufferComplete event when the video player goes into the buffering state and begins to buffer content.
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferComplete);

........

// 5. Call trackComplete() when the playback reaches the end, i.e., when playback completes and finishes playing.
this._mediaHeartbeat.trackComplete();

........

// 6. Call trackSessionEnd() when the playback session is over. This method must be called even if the user does not watch the video to completion.
VOD multiple trackers in parallel

Scenario
In this scenario, there are two sessions running in parallel for two separate videos and using two separate instances of MediaHeartbeat.

This scenario is identical to the VOD playback with no ads scenario, except there are two sessions that are running in parallel for two separate videos. Each of these sessions uses a separate instance of MediaHeartbeat.

Unless specified, the network calls are the same as the VOD playback with no ads scenario.

Parameters

Table 16: Heartbeat session

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:sid</td>
<td>Unique session ID</td>
<td>A unique session ID that exists in all of the heartbeat network calls until the <code>trackSessionEnd</code> method is called.</td>
</tr>
</tbody>
</table>

Sample Code

```java
public class VideoAnalyticsProvider implements MediaHeartbeatDelegate {
   private VideoPlayer _player;
```
private MediaHeartbeat _heartbeat;

public VideoAnalyticsProvider(VideoPlayer player) {
    if (player == null) {
        throw new IllegalArgumentException("Player reference cannot be null.");
    }
    _player = player;
    _player.addObserver(this);

    // Media Heartbeat initialization
    MediaHeartbeatConfig config = new MediaHeartbeatConfig();
    config.trackingServer = HEARTBEAT_TRACKING_SERVER;
    config.channel = HEARTBEAT_CHANNEL;
    config.appVersion = HEARTBEAT_SDK;
    config.ovp = HEARTBEAT_OVP;
    config.playerName = PLAYER_NAME;
    config.ssl = false;
    config.debugLogging = true;

    _heartbeat = new MediaHeartbeat(this, config);
}

@Override
public MediaObject getQoSObject() {
    return MediaHeartbeat.createQoSObject(BITRATE,
                                           STARTUP_TIME,
                                           FPS,
                                           DROPPED_FRAMES);
}

@Override
public Double getCurrentPlaybackTime() {
    return _player.getCurrentPlaybackTime();
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Bootstrap the AdobeMobile library.
    Config.setContext(this.getApplicationContext());

    // Create first VideoPlayer instance.
    _player1 = new VideoPlayer();
    // Create first VideoAnalyticsProvider instance and
    // attach it to the VideoPlayer instance.
    _analyticsProvider1 = new VideoAnalyticsProvider(_player1);

    // Load the main video content.
    Uri uri =
        Uri.parse("android.resource://" + getPackageName() + "+" + R.raw.video1);
    _player1.loadContent(uri);

    // Create second VideoPlayer instance.
    _player2 = new VideoPlayer();
    // Create second VideoAnalyticsProvider instance and
    // attach it to the VideoPlayer instance.
    _analyticsProvider2 = new VideoAnalyticsProvider(_player2);

    // Load the main video content.
    Uri uri =
        Uri.parse("android.resource://" + getPackageName() + "+" + R.raw.video2);
    _player2.loadContent(uri);
}
Both instances of VideoAnalyticsProvider and MediaHeartbeat track two separate sessions, each with its own unique session IDs. The two sessions in the Charles debugging tool or debug logs can be identified by using the session ID value. To display this scenario in Android, set up the following code:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_VAL_1, CUSTOM_KEY_1);
videoMetadata.put(CUSTOM_VAL_2, CUSTOM_KEY_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used, // i.e., there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......
......

// 2. Call trackPlay() when the playback actually starts, i.e., when the first // frame of main content is rendered on the screen.
_mediaHeartbeat.trackPlay();

......
......

// 3. Call trackComplete() when the playback reaches the end, i.e., when the // video completes and finishes playing.
_mediaHeartbeat.trackComplete();

......
......

// 4. Call trackSessionEnd() when the playback session is over. This method must // be called even if the user does not watch the video to completion.
_mediaHeartbeat.trackSessionEnd();

......
......
```

**iOS**

```objective-c
@interface VideoAnalyticsProvider : NSObject <ADBMediaHeartbeatDelegate>
@end

@implementation VideoAnalyticsProvider

VideoPlayer *_player;

- (instancetype)initWithPlayer:(AVPlayer *)player {
    if (self = [super init]) {
        _player = player;

        ADBMediaHeartbeatConfig *config = 
            [[ADBMediaHeartbeatConfig alloc] init];
        config.trackingServer = HEARTBEAT_TRACKING_SERVER;
        config.channel = HEARTBEAT_CHANNEL;
        config.appVersion = HEARTBEAT_SDK_VERSION;
        config.playerName = PLAYER_NAME;
        config.ssl = SSL_SETTING;
        config.debugLogging = DEBUG_SETTING;

        ADBMediaHeartbeatConfig *config =
```
[[ADBMediaHeartbeatConfig alloc] init];
_mediaHeartbeat =
[[ADBMediaHeartbeat alloc] initWithDelegate:self config:config];

[self setupPlayerNotifications];
}
return self;

- (ADBMediaObject *)getQoSInfo {
  return [ADBMediaHeartbeat createQoSObjectWithBitrate:CURRENT_BITRATE_VALUE
    startupTime:CALCULATED_STARTED_TIME
    fps:CALCULATED_FPS
    droppedFrames:DROPPED_FRAMES_COUNT];
}

- (NSTimeInterval)getCurrentPlaybackTime {
  return CMTimeGetSeconds(_player.currentTime);
}
@end

- (void)viewDidAppear:(BOOL)animated {
  [super viewDidAppear:animated];
  [ADBMobile setDebugLogging:YES];

  // Setup the first video player
  NSURL *streamUrl = [NSURL URLWithString:CONTENT_URL_1];
  if (!self.videoPlayer1) {
    self.videoPlayer1 = [[VideoPlayer alloc] initWithContentURL:streamUrl];
    //setup player
  }

  // Create the VideoAnalyticsProvider instance and attach it to the first
  // VideoPlayer instance.
  if (!self.videoAnalyticsProvider1) {
    self.videoAnalyticsProvider1 =
      [[VideoAnalyticsProvider alloc] initWithPlayerDelegate:self.videoPlayer1];
  }

  // Setup the second video player
  NSURL *streamUrl2 = [NSURL URLWithString:CONTENT_URL_2];
  if (!self.videoPlayer2) {
    self.videoPlayer2 = [[VideoPlayer alloc] initWithContentURL:streamUrl2];
    //setup player
  }

  // Create the VideoAnalyticsProvider instance and attach it to the second
  // VideoPlayer instance.
  if (!self.videoAnalyticsProvider2) {
    self.videoAnalyticsProvider2 =
      [[VideoAnalyticsProvider alloc] initWithPlayerDelegate:self.videoPlayer2];
  }
}

- (void)viewDidAppear:(BOOL)animated {
  [super viewDidAppear:animated];
  [ADBMobile setDebugLogging:YES];

  // Setup the first video player
  NSURL *streamUrl = [NSURL URLWithString:CONTENT_URL_1];
  if (!self.videoPlayer1) {
    self.videoPlayer1 = [[VideoPlayer alloc] initWithContentURL:streamUrl];
    //setup player
  }
}
// Create the VideoAnalyticsProvider instance and attach it to the first
// VideoPlayer instance.
if (!self.videoAnalyticsProvider1) {
    self.videoAnalyticsProvider1 = 
        [[VideoAnalyticsProvider alloc] initWithPlayerDelegate:self.videoPlayer1];
}

// Setup the second video player
NSURL *streamUrl2 = [NSURL URLWithString:CONTENT_URL_2];
if (!self.videoPlayer2) {
    self.videoPlayer2 = [[VideoPlayer alloc] initWithContentURL:streamUrl2];
    //setup player
}

// Create the VideoAnalyticsProvider instance and attach it to the second VideoPlayer instance.
if (!self.videoAnalyticsProvider2) {
    self.videoAnalyticsProvider2 = 
        [[VideoAnalyticsProvider alloc] initWithPlayerDelegate:self.videoPlayer2];
}

Both instances of VideoAnalyticsProvider and ADBMediaHeartbeat track two separate sessions, each with its own unique session IDs. The two sessions in the Charles debugging tool or debug logs can be identified by using the session ID value.

To display this scenario in iOS, set up the following code:

// Set up mediaObject
ADBMediaObject *mediaObject = 
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
        length:MEDIA_LENGTH
        streamType:ADBMediaHeartbeatStreamTypeVOD];
NSMutableDictionary *videoContextData = 
    [[[NS MUTABLE DICTIONARY alloc] init];
    [videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
    [videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
//    i.e., there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];
........
........

// 2. Call trackPlay when the playback actually starts, i.e., when the first
// frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];
........
........

// 3. Call trackComplete when the playback reaches the end, i.e., when the
// video completes and finishes playing.
[_mediaHeartbeat trackComplete];
........
........

// 4. Call trackSessionEnd when the playback session is over. This method
// must be called even if the user does not watch the video to completion.
[_mediaHeartbeat trackSessionEnd];
........
........

• JavaScript -

    var MediaHeartbeat = ADB.va.MediaHeartbeat;
    var MediaHeartbeatConfig = ADB.va.MediaHeartbeatConfig;
    var MediaHeartbeatDelegate = ADB.va.MediaHeartbeatDelegate;
function VideoAnalyticsProvider(player) {
    if (!player) {
        throw new Error("Illegal argument. Player reference cannot be null."
    }
    this._player = player;
    // Media Heartbeat initialization
    var mediaConfig = new MediaHeartbeatConfig();
    mediaConfig.trackingServer = Configuration.HEARTBEAT.TRACKING_SERVER;
    mediaConfig.playerName = Configuration.PLAYER.NAME;
    mediaConfig.channel = Configuration.HEARTBEAT.CHANNEL;
    mediaConfig.debugLogging = true;
    mediaConfig.appVersion = Configuration.HEARTBEAT.SDK;
    mediaConfig.ssl = false;
    mediaConfig.ovp = Configuration.HEARTBEAT.OVP;
    var mediaDelegate = new MediaHeartbeatDelegate();
    mediaDelegate.getCurrentPlaybackTime = function() {
        return player.getCurrentPlaybackTime();
    };
    mediaDelegate.prototype.getQoSObject = function() {
        return player.getQoSInfo();
    };
    this._mediaHeartbeat = new MediaHeartbeat(mediaDelegate, mediaConfig, appMeasurement);
}

// Create first VideoPlayer instance.
var _player1 = new VideoPlayer();
// Create the first VideoAnalyticsProvider instance
// and attach it to the VideoPlayer instance.
analyticsProvider1 = new VideoAnalyticsProvider(_player1);

// Load the main video content.
_player1.loadContent(URL_TO_MEDIA_1);

// Create second VideoPlayer instance.
var _player2 = new VideoPlayer();
// Create second VideoAnalyticsProvider instance and
// attach it to the VideoPlayer instance.
analyticsProvider2 = new VideoAnalyticsProvider(_player2);

// Load the main video content for the 2nd player.
_player2.loadContent(URL_TO_MEDIA_2);

Both instances of VideoAnalyticsProvider and MediaHeartbeat track two separate sessions, each with its own unique session IDs. You can see the two sessions in the Charles debugging tool.

**VOD one tracker for multiple sessions**

**Scenario**
In this scenario, the MediaHeartbeat instance is used to create two separate sessions in sequence.

This scenario is the same as the **VOD playback with no ads** scenario.
### Parameters

#### Table 17: Heartbeat Session

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:sid</td>
<td>Unique session ID</td>
<td>A unique session ID that exists in all the heartbeat network calls until <code>trackSessionEnd</code> is called.</td>
</tr>
</tbody>
</table>

### Sample Code

#### Android

To create two instances of `MediaHeartbeat` for two video players, set up the following code:

```java
public class VideoAnalyticsProvider implements MediaHeartbeatDelegate {

    private VideoPlayer _player;
    private MediaHeartbeat _heartbeat;

    public VideoAnalyticsProvider(VideoPlayer player) {
        if (player == null) {
            throw new IllegalArgumentException("Player reference cannot be null.");
        }
        _player = player;
        _player.addObserver(this);
        // Media Heartbeat initialization
        MediaHeartbeatConfig config = new MediaHeartbeatConfig();
        config.trackingServer = HEARTBEAT_TRACKING_SERVER;
        config.channel = HEARTBEAT_CHANNEL;
        config.appVersion = HEARTBEAT_SDK;
        config.ovp = HEARTBEAT_OVP;
        config.playerName = PLAYER_NAME;
        config.ssl = false;
        config.debugLogging = true;
        _heartbeat = new MediaHeartbeat(this, config);
    }

    @Override
    public MediaObject getQoSObject() {
        return MediaHeartbeat.createQoSObject(BITRATE, STARTUP_TIME, FPS, DROPPED_FRAMES);
    }

    @Override
    public Double getCurrentPlaybackTime() {
```
return _player.getCurrentPlaybackTime();
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Bootstrap the AppMeasurement library.
    Config.setContext(this.getApplicationContext());

    // Create the VideoPlayer instance.
    _player = new VideoPlayer();

    // Create the VideoAnalyticsProvider instance and
    // attach it to the VideoPlayer instance.
    _analyticsProvider = new VideoAnalyticsProvider(_player);

    // Load the main video content.
    Uri uri = Uri.parse("android.resource://" + getPackageName() + "/" + R.raw.video1);
    _player.loadContent(uri);
}

To display the first session by using the VideoAnalyticsProvider (hence MediaHeartbeat) instance in Android, set up the following code:

// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_VAL_1, CUSTOM_KEY_1);
videoMetadata.put(CUSTOM_VAL_2, CUSTOM_KEY_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
//    i.e., there is an intent to start playback.
    _mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......
......

// 2. Call trackPlay() when the playback actually starts, i.e., when the first
//    frame of main content is rendered on the screen.
    _mediaHeartbeat.trackPlay();

......
......

// 3. Call trackComplete() when the playback reaches the end, i.e., when the
//    video completes and finishes playing.
    _mediaHeartbeat.trackComplete();

......
......

// 4. Call trackSessionEnd() when the playback session is over. This method must
//    be called even if the user does not watch the video to completion.
    _mediaHeartbeat.trackSessionEnd();

......
......
To display the second session, you can use the same `VideoAnalyticsProvider` (MediaHeartbeat) instance as the first session, but for a new session:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.VOD
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_VAL_1, CUSTOM_KEY_1);
videoMetadata.put(CUSTOM_VAL_2, CUSTOM_KEY_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
//    i.e., there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......

// 2. Call trackPlay() when the playback actually starts, i.e., when the first
//    frame of main content is rendered on the screen.
_mediaHeartbeat.trackPlay();

......

// 3. Call trackComplete() when the playback reaches the end, i.e., when the
//    video completes and finishes playing.
_mediaHeartbeat.trackComplete();

......

// 4. Call trackSessionEnd() when the playback session is over. This method
//    must be called even if the user does not watch the video to completion.
_mediaHeartbeat.trackSessionEnd();

......

• iOS -

To create two instances of MediaHeartbeat for two video players, enter the following:

```objective-c
@interface VideoAnalyticsProvider : NSObject <ADBMediaHeartbeatDelegate>
@end

@implementation VideoAnalyticsProvider

- (instancetype)initWithPlayer:(AVPlayer *)player {
    if (self = [super init]) {
        _player = player;

        ADBMediaHeartbeatConfig *config = [[ADBMediaHeartbeatConfig alloc] init];
        config.trackingServer = HEARTBEAT_TRACKING_SERVER;
        config.channel = HEARTBEAT_CHANNEL;
        config.appVersion = HEARTBEAT_SDK_VERSION;
        config.playerName = PLAYER_NAME;
        config.ssl = SSL_SETTING;
        config.debugLogging = DEBUG_SETTING;

        _mediaHeartbeat = [[ADBMediaHeartbeat alloc] initWithDelegate:self config:config];
    }

    return self;
}
```
[self setupPlayerNotifications];
}
return self;
}

-(ADBMediaObject *)getQoSInfo {
return [ADBMediaHeartbeat createQoSObjectWithBitrate:CURRENT_BITRATE_VALUE
startupTime:CALCULATED_STARTED_TIME
fps:CALCULATED_FPS
droppedFrames:DROPPED_FRAMES_COUNT];
}

-(NSTimeInterval)getCurrentPlaybackTime {
return CMTimeGetSeconds(_player.currentTime);
}
@end

-(void)viewDidAppear:(BOOL)animated {
[super viewDidAppear:animated];
[ADBMobile setDebugLogging:YES];

// Setup the first video player
NSURL *streamUrl = [NSURL URLWithString:CONTENT_URL];
if (!self.videoPlayer) {
    self.videoPlayer = [[VideoPlayer alloc] initWithContentURL:streamUrl];
    //setup player
}

// Create the VideoAnalyticsProvider instance and attach it to the first
// VideoPlayer instance.
if (!self.videoAnalyticsProvider) {
    self.videoAnalyticsProvider =
        [[VideoAnalyticsProvider alloc] initWithPlayerDelegate:self.videoPlayer];
}
}

To display the first session by using the VideoAnalyticsProvider (hence MediaHeartbeat) instance in iOS, set up the following code:

// Set up mediaObject
ADBMediaObject *mediaObject =
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
    length:MEDIA_LENGTH
    streamType:ADBMediaHeartbeatStreamTypeVOD];
NSDictionary *videoContextData =
    [[[NSDictionary alloc] init];
    [videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
    [videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
// i.e., there's an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// 2. Call trackPlay when the playback actually starts, i.e., when the first
// frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];

// 3. Call trackComplete when the playback reaches the end, i.e., when the
// video completes and finishes playing.
[_mediaHeartbeat trackComplete];

4. Call `trackSessionEnd` when the playback session is over. This method must be called even if the user does not watch the video to completion.

```objective-c
[_mediaHeartbeat trackSessionEnd];
```

To display the second session, you can use the same `VideoAnalyticsProvider (MediaHeartbeat)` instance as the first session, but for a new session:

```objective-c
// Set up mediaObject
ADBMediaObject *mediaObject =
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
     length:MEDIA_LENGTH
     streamType:ADBMediaHeartbeatStreamTypeVOD];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call `trackSessionStart` when the user clicks Play or if autoplay is used, i.e., there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// 2. Call `trackPlay` when the playback actually starts, i.e. when the first frame of the main content is rendered on the screen.
[_mediaHeartbeat trackPlay];

// 3. Call `trackComplete` when the playback reaches the end, i.e., when the video completes and finishes playing.
[_mediaHeartbeat trackComplete];

// 4. Call `trackSessionEnd` when the playback session is over. This method must be called even if the user does not watch the video to completion.
[_mediaHeartbeat trackSessionEnd];

• JavaScript -

```javascript
var MediaHeartbeat = ADB.va.MediaHeartbeat;
var MediaHeartbeatConfig = ADB.va.MediaHeartbeatConfig;
var MediaHeartbeatDelegate = ADB.va.MediaHeartbeatDelegate;

function VideoAnalyticsProvider(player) {
    if (!player) {
        throw new Error("Illegal argument. Player reference cannot be null.");
    }
    this._player = player;

    // Media Heartbeat initialization
    var mediaConfig = new MediaHeartbeatConfig();
    mediaConfig.trackingServer = Configuration.HEARTBEAT.TRACKING_SERVER;
    mediaConfig.playerName = Configuration.PLAYER.NAME;
    mediaConfig.channel = Configuration.HEARTBEAT.CHANNEL;
    mediaConfig.debugLogging = true;
    mediaConfig.appVersion = Configuration.HEARTBEAT.SDK;
    mediaConfig.ssl = false;
    mediaConfig.ovp = Configuration.HEARTBEAT.OVP;
    var mediaDelegate = new MediaHeartbeatDelegate();
```
mediaDelegate.getCurrentPlaybackTime = function() {
    return player.getCurrentPlaybackTime();
};

mediaDelegate.prototype.getQoSObject = function() {
    return player.getQoSInfo();
};

this._mediaHeartbeat = new MediaHeartbeat(mediaDelegate, mediaConfig, appMeasurement);

// Create the first VideoPlayer instance.
var _player1 = new VideoPlayer();

// Create the first VideoAnalyticsProvider instance and
// attach it to the VideoPlayer instance.
analyticsProvider1 = new VideoAnalyticsProvider(_player1);

// Load the main content.
_player1.loadContent(URL_TO_MEDIA_1);

## Live main content

### Scenario

In this scenario, there is one live asset with no ads played for 40 secs after joining the live stream.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks Play</td>
<td>trackSessionStart</td>
<td>Analytics Content Start, Heartbeat Content Start</td>
<td>This can be a user clicking Play or an auto-play event.</td>
</tr>
<tr>
<td>The first frame of the video plays.</td>
<td>trackPlay</td>
<td>Heartbeat Content Play</td>
<td>This method triggers the timer. Heartbeats are sent every 10 seconds as long as playback continues.</td>
</tr>
<tr>
<td>The content plays.</td>
<td></td>
<td>Content Heartbeats</td>
<td></td>
</tr>
<tr>
<td>The session is over.</td>
<td>trackSessionEnd</td>
<td></td>
<td>SessionEnd means the end of a viewing session. This API must be called even if the user does not watch the video to completion.</td>
</tr>
</tbody>
</table>

### Parameters

Many of the same values that you see on Adobe Analytics Content Start Calls you will also see on Heartbeat Content Start Calls. You will also see lots of other parameters that Adobe uses to populate the various Video reports in Adobe Analytics. We won't be covering all of them here, just the really important ones.

**Table 18: Heartbeat Content Start**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:sc:rsid</td>
<td>&lt;Your Adobe Report Suite ID&gt;</td>
<td></td>
</tr>
<tr>
<td>s:sc:tracking_serve</td>
<td>&lt;Your Analytics Tracking Server URL&gt;</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>s:user:mid</td>
<td>s:user:mid</td>
<td>Should match the mid value on the Adobe Analytics Content Start Call</td>
</tr>
<tr>
<td>s:event:type</td>
<td>&quot;start&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;main&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:video_id</td>
<td>&lt;Your Video Name&gt;</td>
<td></td>
</tr>
<tr>
<td>s:stream:type</td>
<td>live</td>
<td></td>
</tr>
<tr>
<td>s:meta:*</td>
<td>optional</td>
<td>Custom metadata set on the video</td>
</tr>
</tbody>
</table>

**Content Heartbeats**

During video playback, there is a timer that will send one or more heartbeats every 10 seconds. These heartbeats will contain information about playback, ads, buffering, and a number of other things. The exact content of each heartbeat is beyond the scope of this document, the critical thing to validate is that heartbeats are being triggered consistently while playback continues.

In the content heartbeats, look for a few specific things:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;play&quot;</td>
<td></td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>&lt;playhead position&gt; e.g., 50, 60, 70</td>
<td>This should reflect the current position of the playhead.</td>
</tr>
</tbody>
</table>

**Heartbeat Content Complete**

There will not be a complete call, because the live stream was never completed.

**Sample Code**

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.LIVE
);
```

**Android**

Here is the expected API call order:

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.LIVE
);
```
HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_VAL_1, CUSTOM_KEY_1);
videoMetadata.put(CUSTOM_VAL_2, CUSTOM_KEY_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used, 
// i.e., there is an intent to start playback. 
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

// 2. Call trackPlay() when the playback actually starts, i.e., when the first 
// frame of main content is rendered on the screen. 
_mediaHeartbeat.trackPlay();

// 3. Call trackSessionEnd() when user ends the playback session. Since the user 
// does not watch live video to completion, there 
// is no need to call trackComplete. 
_mediaHeartbeat.trackSessionEnd();

• iOS -

Here is the expected API call order:

// Set up mediaObject
ADBMediaObject *mediaObject = 
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME 
    length:MEDIA_LENGTH 
    streamType:ADBMediaHeartbeatStreamTypeLIVE];

NSMutableDictionary *videoContextData = 
    [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used, 
// i.e., there is an intent to start playback. 
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// 2. Call trackPlay when the playback actually starts, i.e., when the first 
// frame of the main content is rendered on the screen. 
[_mediaHeartbeat trackPlay];

// 3. Call trackSessionEnd when user ends the playback session. Since the user 
// does not watch live video to completion, there is no need to call 
// trackComplete. 
[_mediaHeartbeat trackSessionEnd];

• JavaScript -

Here is the expected API call order:

// Set up mediaObject
var mediaInfo = 
    MediaHeartbeat.createMediaObject(Configuration.MEDIA_NAME, 
    Configuration.MEDIA_ID, 
    Configuration.MEDIA_LENGTH, 
    MediaHeartbeat.StreamType.VOD);
```javascript
var videoMetadata = {
    CUSTOM_KEY_1: CUSTOM_VAL_1,
    CUSTOM_KEY_2: CUSTOM_VAL_2,
    CUSTOM_KEY_3: CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay is used, i.e., there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......

// 2. Call trackPlay() when the playback actually starts, i.e., when the first frame of video is rendered on the screen.
this._mediaHeartbeat.trackPlay();

......

// 3. Call trackSessionEnd() when user ends the playback session.
// Since user does not watch live video to completion, there is no need to call trackComplete().
this._mediaHeartbeat.trackSessionEnd();

......

Live main content with sequential tracking

Scenario

In this scenario, there is one live asset with no ads played for 40 secs after joining the live stream.

This is the same scenario as the VOD playback with no ads scenario, but a part of the content is scrubbed through and a seek is completed from one point in main content to another point.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Heartbeat method</th>
<th>Network calls</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>User clicks Play</td>
<td>trackSessionStart</td>
<td>Analytics Content Start, Heartbeat Content Start</td>
<td>The measurement library is unaware that there is a pre-roll ad, so these network calls are identical to the VOD playback with no ads scenario.</td>
</tr>
<tr>
<td>First frame of the content plays.</td>
<td>trackPlay</td>
<td>Heartbeat Content Play</td>
<td>When chapter content plays before main content, the Heartbeats start when the chapter starts.</td>
</tr>
<tr>
<td>Content plays</td>
<td></td>
<td>Content Heartbeats</td>
<td>This network call is exactly the same as the VOD playback with no ads scenario.</td>
</tr>
<tr>
<td>Session1 Over (Episode1 ended)</td>
<td>trackComplete</td>
<td>Heartbeat Content Complete</td>
<td>Complete means session1 for 1st episode was reached and watched completely. Before starting session for next</td>
</tr>
<tr>
<td></td>
<td>trackSessionEnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trigger</td>
<td>Heartbeat method</td>
<td>Network calls</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
<td>---------------</td>
<td>-------</td>
</tr>
<tr>
<td>Episode2 started (Session2 start)</td>
<td>trackSessionStart</td>
<td>Analytics Content Start Heartbeat Content Start</td>
<td>This is because user watched first episode and continued watching into another episode</td>
</tr>
<tr>
<td>1st Frame of Video</td>
<td>trackPlay</td>
<td>Heartbeat Content Play</td>
<td>This method triggers the timer and from this point forward, heartbeats will be sent every 10 seconds as long as playback continues.</td>
</tr>
<tr>
<td>Content Plays</td>
<td></td>
<td>Content Heartbeats</td>
<td></td>
</tr>
<tr>
<td>Session Over (Episode2 ended)</td>
<td>trackComplete trackSessionEnd</td>
<td>Heartbeat Content Complete</td>
<td>Complete means session2 for 2nd episode was reached and watched completely. Before starting session for next episode this session must be ended.</td>
</tr>
</tbody>
</table>

### Parameters

**Table 19: Heartbeat Content Start**

Many of the same values that you see on Adobe Analytics Content Start Calls you will also see on Heartbeat Content Start Calls. You will also see lots of other parameters that Adobe uses to populate the various Video reports in Adobe Analytics. We won't be covering all of them here, just the really important ones.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:sc:rsid</td>
<td>&lt;Your Adobe Report Suite ID&gt;</td>
<td></td>
</tr>
<tr>
<td>s:sc:tracking_server</td>
<td>&lt;Your Analytics Tracking Server URL&gt;</td>
<td>Should match the mid value on the Adobe Analytics Content Start Call</td>
</tr>
<tr>
<td>s:user:mid</td>
<td>s:user:mid</td>
<td></td>
</tr>
<tr>
<td>s:event:type</td>
<td>&quot;start&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;main&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:video_id</td>
<td>&lt;Your Video Name&gt;</td>
<td></td>
</tr>
<tr>
<td>s:stream:type</td>
<td>live</td>
<td></td>
</tr>
<tr>
<td>s:meta:*</td>
<td>optional</td>
<td>Custom metadata set on the video</td>
</tr>
</tbody>
</table>

### Heartbeat Content Play

This should look almost exactly like the Heartbeat Content Start call, but with the key difference in the "s:event:type" parameter. All parameters should still be in place here.
During video playback, there is a timer that will send one or more heartbeats every 10 seconds. These heartbeats will contain information about playback, ads, buffering, and a number of other things. The exact content of each heartbeat is beyond the scope of this document, the critical thing to validate is that heartbeats are being triggered consistently while playback continues.

In the content heartbeats, look for a few specific things:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;play&quot;</td>
<td></td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>&lt;playhead position&gt; e.g., 50, 60, 70</td>
<td>This should reflect the current position of the playhead.</td>
</tr>
</tbody>
</table>

When playback for any given episode has completed (playhead crosses episode boundary), a Heartbeat Content Complete call is sent. This looks like other Heartbeat Calls, but will contain a couple specific things:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>&quot;complete&quot;</td>
<td></td>
</tr>
<tr>
<td>s:asset:type</td>
<td>&quot;main&quot;</td>
<td></td>
</tr>
</tbody>
</table>

**Sample Code**

```java
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.LIVE
);
```
HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_VAL_1, CUSTOM_KEY_1);
videoMetadata.put(CUSTOM_VAL_2, CUSTOM_KEY_2);

// 1. Call trackSessionStart() when the user clicks Play or if autoplay is used,
// i.e., when there is an intent to start playback.
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......

......

// 2. Call trackPlay() when the playback actually starts, i.e., when the first
// frame of main content is rendered on the screen.
_mediaHeartbeat.trackPlay();

......

......

// 3. Call trackComplete() when the playback reaches the end of session,
// i.e., when the video completes and finishes playing last episode/session.
_mediaHeartbeat.trackComplete();

......

......

// 4. Call trackSessionEnd() to end session 1
_mediaHeartbeat.trackSessionEnd();

......

......

// Start tracking session 2 /episode 2 of the same live stream.
// There is no need to reinstantiate a mediaHeartbeat instance for tracking session 2.
// Set up mediaObject
MediaObject mediaInfo = MediaHeartbeat.createMediaObject(
    Configuration.MEDIA_NAME,
    Configuration.MEDIA_ID,
    Configuration.MEDIA_LENGTH,
    MediaHeartbeat.StreamType.LIVE
);

HashMap<String, String> videoMetadata = new HashMap<String, String>();
videoMetadata.put(CUSTOM_VAL_1, CUSTOM_KEY_1);
videoMetadata.put(CUSTOM_VAL_2, CUSTOM_KEY_2);

// 5. Call trackSessionStart() when the playhead reaches a point that denotes the
// start of session 2
_mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);

......

......

// 6. Call trackPlay() to start tracking session 2 playback
_mediaHeartbeat.trackPlay();

......

......

// 7. Call trackComplete() when the playback reaches the end of session 2,
// i.e., the video completes and finishes playing.
_mediaHeartbeat.trackComplete();

......

......

// 8. Call trackSessionEnd() to end session 2
_mediaHeartbeat.trackSessionEnd();
// Continue similarly tracking further sessions in the live stream if required

• iOS -

Here is the expected API call order:

// Set up mediaObject
ADBMediaObject *mediaObject =
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
    length:MEDIA_LENGTH
    streamType:ADBMediaHeartbeatStreamTypeLIVE];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 1. Call trackSessionStart when the user clicks Play or if autoplay is used,
// i.e., there is an intent to start playback.
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// 2. Call trackPlay when the playback actually starts, i.e., when the first
// frame of main content is rendered on the screen.
[_mediaHeartbeat trackPlay];

// 3. Call trackComplete when the playback reaches the end of session,
// i.e., when the video completes and finishes playing the first
// episode/session.
[_mediaHeartbeat trackComplete];

// 4. Call trackSessionEnd to end session 1
[_mediaHeartbeat trackSessionEnd];

// Start tracking session 2 / episode 2 of the same live stream, No need to
// reinstantiate mediaHeartbeat instance for tracking sesison 2.

// Set up mediaObject
ADBMediaObject *mediaObject =
    [ADBMediaHeartbeat createMediaObjectWithName:MEDIA_NAME
    length:MEDIA_LENGTH
    streamType:ADBMediaHeartbeatStreamTypeLIVE];

NSMutableDictionary *videoContextData = [[NSMutableDictionary alloc] init];
[videoContextData setObject:CUSTOM_VAL_1 forKey:CUSTOM_KEY_1];
[videoContextData setObject:CUSTOM_VAL_2 forKey:CUSTOM_KEY_2];

// 5. Call trackSessionStart when the playhead reaches a point that denotes
// start of session 2
[_mediaHeartbeat trackSessionStart:mediaObject data:videoContextData];

// 6. Call trackPlay to start tracking session 2 playback
[_mediaHeartbeat trackPlay];

// 7. Call trackComplete when the playback reaches the end of session 2,
// i.e., when the video completes and finishes playing.
// 8. Call trackSessionEnd to end the session 2
[mediaHeartbeat trackSessionEnd];
..........  

// Continue tracking further sessions in live stream similarly if required

• JavaScript -

Here is the expected API call order:

// Set up mediaObject

var mediaInfo = MediaHeartbeat.createMediaObject(
  Configuration.MEDIA_NAME,
  Configuration.MEDIA_ID,
  Configuration.MEDIA_LENGTH,
  MediaHeartbeat.StreamType.VOD
);

var videoMetadata = {
  CUSTOM_KEY_1 : CUSTOM_VAL_1,
  CUSTOM_KEY_2 : CUSTOM_VAL_2,
  CUSTOM_KEY_3 : CUSTOM_VAL_3
};

// 1. Call trackSessionStart() when Play is clicked or if autoplay is used,
//    i.e., there's an intent to start playback.
this._mediaHeartbeat.trackSessionStart(mediaInfo, videoMetadata);
 ..........  

// 2. Call trackPlay() when the playback actually starts, i.e., when the
//    first frame of video is rendered on the screen.
this._mediaHeartbeat.trackPlay();
 ..........  

// 3. Call trackComplete() when the playback reaches the end of a session,
//    i.e., when playback completes and finishes playing the 1st episode/session.
this._mediaHeartbeat.trackComplete();
 ..........  

// 4. Call trackSessionEnd() to end session 1
this._mediaHeartbeat.trackSessionEnd();
 ..........  

// Start tracking session 2/episode 2 of the same live stream. There is no need
// to reinstantiate a mediaHeartbeat instance for tracking session 2.

// Set up mediaObject
var mediaInfo2 = MediaHeartbeat.createMediaObject(
  Configuration.MEDIA_NAME,
  Configuration.MEDIA_ID,
  Configuration.MEDIA_LENGTH,
  MediaHeartbeat.StreamType.LIVE
);

var videoMetadata2 = {

// 5. Call trackSessionStart() when the playhead reaches a point that denotes
//    the start of session 2
this._mediaHeartbeat.trackSessionStart(mediaInfo2, videoMetadata2);

       ......

// 6. Call trackPlay() to start tracking session 2 playback
this._mediaHeartbeat.trackPlay();

       ......

// 7. Call trackComplete() when the playback reaches the end of session 2,
//    i.e., playback completes and finishes playing.
this._mediaHeartbeat.trackComplete();

       ......

// 8. Call trackSessionEnd() to end session 2
this._mediaHeartbeat.trackSessionEnd();

       ......

// Continue tracking further sessions in live stream similarly if required

Validation

Validation overview

Validation includes a list of required test cases that customers must run through (using Adobe Debug), to submit
logs validating a proper implementation. This section also includes expected results for each event and metadata
parameter.

Test 1: Standard playback

This test case is required as part of the certification request form. It validates general playback and sequencing.

Download the certification request form here: Certification request form.

Video implementations are composed of the following types of tracking calls:

• Video and Ad Start calls are sent directly to the AppMeasurement server.
• Media Analytics (MA) heartbeat calls are sent on start and every ten seconds to the Adobe MA tracking server.

Video tracking will behave the same across all platforms, desktop and mobile.

You must complete and record the actions in the following order:

1. Load the page or app.

   Tracking Servers (For all website and mobile apps):
• **AppMeasurement (Adobe Analytics)** - An RDC tracking server or CNAME that resolves to an RDC server is required for the Experience Cloud Visitor ID service. The analytics tracking server should end in .sc.omtrdc.net or be a CNAME.

• **Media Analytics (Heartbeats)** - This server always has the format [namespace].hb.omtrdc.net, where [namespace] is defined by your login company and is provided by Adobe.

You need to validate certain key, universal variables across all tracking calls.

• **ADOBE**

Adobe Visitor ID (mid): The mid variable is used to capture the value set in the AMCV cookie. The mid variable is the primary identification value for both websites and mobile apps, and also indicates that the Experience Cloud Visitor ID service is set-up properly. It is found in both AppMeasurement and Media Analytics (MA) calls.

<table>
<thead>
<tr>
<th>Table 20: Lifecycle Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>pev2</td>
</tr>
<tr>
<td>mid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 21: Website Page Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>mid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 22: Media Analytics Start Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>pev2</td>
</tr>
<tr>
<td>mid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 23: Heartbeat Play Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>s:event:type</td>
</tr>
<tr>
<td>s:user:mid</td>
</tr>
</tbody>
</table>

💡 **Note**: On MA Start Calls (s:event:type=start) the mid values may not be present. This is OK. They may not appear until the MA Play Calls (s:event:type=play).

<table>
<thead>
<tr>
<th>Table 24: MA Start Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>pev2</td>
</tr>
</tbody>
</table>
2. **Start the video player.**

When the video player starts, the key calls are sent in the following order:

1. Video analytics start*
2. Heartbeat start*
3. Heartbeat analytics start

*These calls contain additional metadata and variables. For call parameters and metadata, see [Start the video player in Test Call Details](#).

Also see your platform's Implementation instructions for additional information about each call.

3. **View ad break if available.**

   - **Ad Start**

     When the video ad starts, the following key calls are sent in the following order:

     1. Video ad analytics start*
     2. Heartbeat ad start*
     3. Heartbeat ad analytics start

     *These calls contain additional metadata and variables. For call parameters and metadata, see [View ad playback in Test Call Details](#).

     Also see your platform's Implementation instructions for additional information about these Ad calls.

   - **Ad Play**

     During ad playback, Heartbeat calls are sent to the Heartbeat server every second.

   - **Ad Complete**

     At the 100% point on a video ad, a Heartbeat complete call will be sent.

4. **Pause ad playback for 30 seconds, if available.**

   - **Ad Pause**

     During ad pause, Heartbeat calls are sent to the Heartbeat server every second.

     ![Note:](#) The playhead value should remain constant during the pause.

5. **Play main content video for 10 minutes uninterrupted.**

   - **Content Play**

     During regular main content playback, Heartbeat calls are sent to the Heartbeat server every ten seconds.

     **Notes:**

     - The playhead position should increment by 10 with every play call.

---

### Table 25: Heartbeat Start Call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>start</td>
</tr>
</tbody>
</table>
• The \texttt{event:duration} value represents the number of milliseconds since the last tracking call and should be roughly the same value on each 10 second call.

For call parameters and metadata, see Play main content in Test Call Details

Also see your platform's Implementation instructions for additional information about these Ad calls.

6. **Pause during playback for at least 30 seconds.**

   On pause of the video player, pause event calls will be sent every 10 seconds. After pause ends the play events should resume.

7. **Seek/scrub video.**

   On scrubbing of video playhead, no special tracking calls are sent, however, when video playback resumes after scrubbing the playhead value should reflect the new position within the main content.

8. **Replay video (VOD only).**

   When a video is replayed, a new set of video start calls should be sent, as if this is a fresh video view.

9. **View next video in playlist.**

   On video start of the next video in a playlist, a new set of video start calls should be sent.

10. **Switch video or stream.**

    When switching live streams, a Heartbeat complete call for the first stream should not be sent. The video start calls and video play calls should begin with the new show and stream name and with the correct playhead and duration values for the new show.

**Test 2: Video interruption**

This test case is required as part of the certification request form and validates mobile interruption behavior.

To download the certification request form, click Certification Request Form.

You must complete and record these tasks in following order:

1. **Start the video player.**

   When the video player starts, the following calls are sent in the following order:

   1. Video analytics start*
   2. Heartbeat start*
   3. Heartbeat analytics start

   *These calls contain additional metadata and variables. For call parameters and metadata, see Start the video player in Test Call Details.

   Also see your platform's Implementation instructions for additional information about each call.

2. **Play main content video for at least 5 minutes uninterrupted.**

   **Content Play**

   During regular main content playback, Heartbeat calls are sent to the Heartbeat server every ten seconds.

   For call parameters and metadata, see Play main content in Test Call Details.
Also see your platform's *Implementation* instructions for additional information about these Ad calls.

3. **Move app or browser to the background.**
   While the app runs in the background, only main:pause calls should be sent to the Heartbeat server, starting with VHL version 1.6.6 and later.

4. **Bring app or browser to background.**
   On returning from background, content playback should resume.

5. **Play main content video for at least 5 minutes uninterrupted.**
   For call parameters and metadata, see *Play main content* in *Test Call Details*.

6. **Close video player.**
   No additional tracking calls should fire after video player is closed.

**Test call details**

This topic presents detailed call information for the validation test topics.

- *Start the video player*
- *View ad playback*
- *Play main content*

### Start the video player

#### Table 26: Media Analytics start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pev2</td>
<td>ms_s</td>
</tr>
<tr>
<td>a.media.friendlyName</td>
<td>Episode Title</td>
</tr>
<tr>
<td>a.media.name</td>
<td>123456</td>
</tr>
<tr>
<td>a.media.length</td>
<td>120</td>
</tr>
<tr>
<td>a.media.playerName</td>
<td>HTML5</td>
</tr>
<tr>
<td>a.media.view</td>
<td>true</td>
</tr>
<tr>
<td>a.contentType</td>
<td>vod</td>
</tr>
<tr>
<td>custom.[value]</td>
<td>Custom metadata fields</td>
</tr>
<tr>
<td>a.media.[value]</td>
<td>Standard metadata fields</td>
</tr>
</tbody>
</table>

**Notes:**

- Additional context data variables should be present and contain metadata. See metadata details below.
- Length for linear streams should be set to the best estimate for the current show.
Table 27: Standard metadata in Media Analytics start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.media.show</td>
<td>Show Title</td>
</tr>
<tr>
<td>a.media.season</td>
<td>6</td>
</tr>
<tr>
<td>a.media.episode</td>
<td>Episode Title</td>
</tr>
<tr>
<td>a.media.asset_id</td>
<td>123456</td>
</tr>
<tr>
<td>a.media.genre</td>
<td>comedy</td>
</tr>
<tr>
<td>a.media.first_air_date</td>
<td>2016-07-04</td>
</tr>
<tr>
<td>a.media.rating</td>
<td>TV-14</td>
</tr>
<tr>
<td>a.media.originator</td>
<td>production house</td>
</tr>
<tr>
<td>a.media.network</td>
<td>network</td>
</tr>
<tr>
<td>a.media.ad_load</td>
<td>1</td>
</tr>
<tr>
<td>a.media.mvpd</td>
<td>mvpd</td>
</tr>
<tr>
<td>a.media.authorized</td>
<td>unlocked</td>
</tr>
<tr>
<td>a.media.feed</td>
<td>no feed</td>
</tr>
<tr>
<td>a.media.stream_format</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 28: Video metadata in Media Analytics start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom.metadataA</td>
<td>value</td>
</tr>
<tr>
<td>custom.metadataB</td>
<td>value</td>
</tr>
</tbody>
</table>

Table 29: Heartbeat start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>start</td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>0</td>
</tr>
<tr>
<td>l:event:duration</td>
<td>4</td>
</tr>
<tr>
<td>l:asset:name</td>
<td>Episode Title</td>
</tr>
<tr>
<td>l:asset:video_id</td>
<td>123456</td>
</tr>
<tr>
<td>l:asset:length</td>
<td>120</td>
</tr>
<tr>
<td>l:stream:type</td>
<td>vod</td>
</tr>
<tr>
<td>s:asset:type</td>
<td>main</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value (sample)</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>s:meta:custom.[value]</td>
<td>Custom metadata fields</td>
</tr>
<tr>
<td>s:meta:a.media.[value]</td>
<td>Standard metadata fields</td>
</tr>
</tbody>
</table>

Notes:
- Additional context data variables should be present and contain metadata. See metadata details below.
- Playhead position for linear streams on video start should be set to the seconds elapsed since the start of the current show, not 0.

### Table 30: Standard metadata in Heartbeat start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:meta:a.media.show</td>
<td>Show</td>
</tr>
<tr>
<td>s:meta:a.media.season</td>
<td>6</td>
</tr>
<tr>
<td>s:meta:a.media.episode</td>
<td>Episode Title</td>
</tr>
<tr>
<td>s:meta:a.media.asset_id</td>
<td>123456</td>
</tr>
<tr>
<td>s:meta:a.media.genre</td>
<td>comedy</td>
</tr>
<tr>
<td>s:meta:a.media.first_air_date</td>
<td>2018-07-04</td>
</tr>
<tr>
<td>s:meta:a.media.rating</td>
<td>TV-14</td>
</tr>
<tr>
<td>s:meta:a.media.originator</td>
<td>production house</td>
</tr>
<tr>
<td>s:meta:a.media.network</td>
<td>network</td>
</tr>
<tr>
<td>s:meta:a.media.ad_load</td>
<td>1</td>
</tr>
<tr>
<td>s:meta:a.media.mvpd</td>
<td>mvpd</td>
</tr>
<tr>
<td>s:meta:a.media.authorized</td>
<td>unlocked</td>
</tr>
<tr>
<td>s:meta:a.media.feed</td>
<td>no feed</td>
</tr>
<tr>
<td>s:meta:a.media.stream_format</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 31: Video metadata in Heartbeat start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:meta:custom.metadata</td>
<td>value</td>
</tr>
<tr>
<td>s:meta:custom.metadata</td>
<td>value</td>
</tr>
</tbody>
</table>

### Table 32: Heartbeat Analytics start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>aa_start</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:event:playhead</td>
<td>0</td>
</tr>
<tr>
<td>1:event:duration</td>
<td>4</td>
</tr>
<tr>
<td>1:asset:name</td>
<td>Episode Title</td>
</tr>
<tr>
<td>1:asset:video_id</td>
<td>123456</td>
</tr>
<tr>
<td>1:asset:length</td>
<td>120</td>
</tr>
<tr>
<td>1:stream:type</td>
<td>vod</td>
</tr>
<tr>
<td>s:asset:type</td>
<td>main</td>
</tr>
</tbody>
</table>

### Notes:

- This call indicates that the heartbeat library has requested that an analytics `pev2=ms_s` call be sent to the analytics server.
- This call does not contain custom metadata.

### View ad playback

#### Table 33: Media Analytics ad start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pev2</td>
<td>msa_s</td>
</tr>
<tr>
<td>a.media.name</td>
<td>123456</td>
</tr>
<tr>
<td>a.media.ad.name</td>
<td>9378</td>
</tr>
<tr>
<td>a.media.ad.friendlyName</td>
<td>Video_VPAID_DFA</td>
</tr>
<tr>
<td>a.media.ad.podFriendlyName</td>
<td>preroll</td>
</tr>
<tr>
<td>a.media.ad.length</td>
<td>15</td>
</tr>
<tr>
<td>a.media.ad.playerName</td>
<td>HTML5</td>
</tr>
<tr>
<td>a.media.ad.pod</td>
<td>c27aaf3ff8224bb6b9ebfe1b2e79073d_1</td>
</tr>
<tr>
<td>a.media.ad.podPosition</td>
<td>1</td>
</tr>
<tr>
<td>a.media.ad.podSecond</td>
<td>0.0</td>
</tr>
<tr>
<td>a.media.ad.view</td>
<td>True</td>
</tr>
<tr>
<td>custom.[value]</td>
<td>Metadata fields</td>
</tr>
<tr>
<td>a.media.[value]</td>
<td>Standard metadata fields</td>
</tr>
</tbody>
</table>

**Note:** Additional context data variables should be present and contain metadata. See metadata details below.
Table 34: Standard metadata in Media Analytics ad start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.media.show</td>
<td>Show Title</td>
</tr>
<tr>
<td>a.media.season</td>
<td>6</td>
</tr>
<tr>
<td>a.media.episode</td>
<td>Episode Title</td>
</tr>
<tr>
<td>a.media.asset_id</td>
<td>123456</td>
</tr>
<tr>
<td>a.media.genre</td>
<td>comedy</td>
</tr>
<tr>
<td>a.media.first_air_date</td>
<td>2016-07-04</td>
</tr>
<tr>
<td>a.media.rating</td>
<td>TV-14</td>
</tr>
<tr>
<td>a.media.originator</td>
<td>production house</td>
</tr>
<tr>
<td>a.media.network</td>
<td>network</td>
</tr>
<tr>
<td>a.media.ad_load</td>
<td>1</td>
</tr>
<tr>
<td>a.media.mvpd</td>
<td>mvpd</td>
</tr>
<tr>
<td>a.media.authorized</td>
<td>unlocked</td>
</tr>
<tr>
<td>a.media.feed</td>
<td>no feed</td>
</tr>
<tr>
<td>a.media.stream_format</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 35: Video metadata in Media Analytics ad start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom.metadata</td>
<td>value</td>
</tr>
<tr>
<td>custom.metadata</td>
<td>value</td>
</tr>
</tbody>
</table>

Table 36: Heartbeat ad start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>start</td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>0</td>
</tr>
<tr>
<td>l:event:duration</td>
<td>4</td>
</tr>
<tr>
<td>l:asset:ad_id</td>
<td>9378</td>
</tr>
<tr>
<td>l:asset:length</td>
<td>120</td>
</tr>
<tr>
<td>l:stream:type</td>
<td>vod</td>
</tr>
<tr>
<td>s:asset:type</td>
<td>ad</td>
</tr>
<tr>
<td>s:meta:custom.[value]</td>
<td>Custom metadata fields</td>
</tr>
</tbody>
</table>
Notes:
• Additional context data variables should be present and contain metadata. See metadata details below.
• Ad length may be set to -1 if not available on ad start.

Table 37: Standard metadata in Heartbeat ad start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:meta:a.media.show</td>
<td>Show</td>
</tr>
<tr>
<td>s:meta:a.media.season</td>
<td>6</td>
</tr>
<tr>
<td>s:meta:a.media.episode</td>
<td>Episode Title</td>
</tr>
<tr>
<td>s:meta:a.media.asset_id</td>
<td>123456</td>
</tr>
<tr>
<td>s:meta:a.media.genre</td>
<td>comedy</td>
</tr>
<tr>
<td>s:meta:a.media.first_air_date</td>
<td>2018-07-04</td>
</tr>
<tr>
<td>s:meta:a.media.rating</td>
<td>TV-14</td>
</tr>
<tr>
<td>s:meta:a.media.originator</td>
<td>production house</td>
</tr>
<tr>
<td>s:meta:a.media.network</td>
<td>network</td>
</tr>
<tr>
<td>s:meta:a.media.ad_load</td>
<td>1</td>
</tr>
<tr>
<td>s:meta:a.media.mvpd</td>
<td>mvpd</td>
</tr>
<tr>
<td>s:meta:a.media.authorized</td>
<td>unlocked</td>
</tr>
<tr>
<td>s:meta:a.media.feed</td>
<td>no feed</td>
</tr>
<tr>
<td>s:meta:a.media.stream_format</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 38: Video metadata in Heartbeat ad start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:meta:custom.metadata</td>
<td>value</td>
</tr>
<tr>
<td>s:meta:custom.metadata</td>
<td>value</td>
</tr>
</tbody>
</table>

Table 39: Heartbeat Analytics ad start call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>aa_ad_start</td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>0</td>
</tr>
<tr>
<td>l:event:duration</td>
<td>0</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value (sample)</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>l:asset:ad_id</td>
<td>9378</td>
</tr>
<tr>
<td>l:asset:ad_length</td>
<td>15</td>
</tr>
<tr>
<td>l:stream:type</td>
<td>vod</td>
</tr>
<tr>
<td>s:asset:type</td>
<td>ad</td>
</tr>
</tbody>
</table>

Notes:
- Additional context data variables should be present and contain metadata. See metadata details below.
- Ad length may be set to -1 if not available on ad start.

Table 40: Heartbeat ad play call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>play</td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>15</td>
</tr>
<tr>
<td>l:event:duration</td>
<td>0</td>
</tr>
<tr>
<td>l:asset:ad_id</td>
<td>9378</td>
</tr>
<tr>
<td>l:asset:ad_length</td>
<td>15</td>
</tr>
<tr>
<td>l:stream:type</td>
<td>vod</td>
</tr>
<tr>
<td>s:asset:type</td>
<td>ad</td>
</tr>
</tbody>
</table>

Table 41: Heartbeat ad complete call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>complete</td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>15</td>
</tr>
<tr>
<td>l:event:duration</td>
<td>0</td>
</tr>
<tr>
<td>l:asset:ad_id</td>
<td>9378</td>
</tr>
<tr>
<td>l:asset:ad_length</td>
<td>15</td>
</tr>
<tr>
<td>l:stream:type</td>
<td>vod</td>
</tr>
<tr>
<td>s:asset:type</td>
<td>ad</td>
</tr>
</tbody>
</table>
Play main content

Table 42: Heartbeat play call

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type</td>
<td>play</td>
</tr>
<tr>
<td>l:event:playhead</td>
<td>29</td>
</tr>
<tr>
<td>l:event:duration</td>
<td>10189</td>
</tr>
<tr>
<td>l:asset:name</td>
<td>Episode Title</td>
</tr>
<tr>
<td>l:asset:video_id</td>
<td>123456</td>
</tr>
<tr>
<td>l:asset:length</td>
<td>120</td>
</tr>
<tr>
<td>l:stream:type</td>
<td>vod</td>
</tr>
<tr>
<td>s:asset:type</td>
<td>main</td>
</tr>
</tbody>
</table>

Notes:
- The playhead position should increment by 10 with every play call.
- The l:event:duration value represents the number of milliseconds since the last tracking call and should be roughly the same value on each 10 second call.

Heartbeat parameter descriptions

List of heartbeat parameters that Adobe collects and processes on the heartbeats server:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required / Optional</th>
<th>Data Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Events</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s:event:type</td>
<td>R</td>
<td>Media SDK</td>
<td>The type of the event being tracked. Event types:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=start</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=complete</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=chapter_start</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=chapter_complete</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=buffer</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=pause</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=resume</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=bitrate_change</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=aa_start</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=stall</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>s:event:type=end</em></td>
</tr>
<tr>
<td>l:event:prev_ts</td>
<td>R</td>
<td>Media SDK</td>
<td>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 media session.</td>
</tr>
<tr>
<td>l:event:ts</td>
<td>R</td>
<td>Media SDK</td>
<td>The timestamp of the event.</td>
</tr>
<tr>
<td>Name</td>
<td>Required / Optional</td>
<td>Data Source</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1:event:duration</td>
<td>R</td>
<td>Media SDK</td>
<td>This value is set internally (in milliseconds) by the VHL Library, not by the player. It is used to compute the time spent metrics on the backend. For example a.media.totalTimePlayed is computed as a sum of the duration for all the Play (type=play) heartbeats that are generated.</td>
</tr>
<tr>
<td>s:event:playhead</td>
<td>R</td>
<td>VideoInfo object</td>
<td>The playhead was inside the currently active asset (main or ad), when the event was recorded.</td>
</tr>
<tr>
<td>s:event:sid</td>
<td>R</td>
<td>Media SDK</td>
<td>The session ID (a randomly generated string). All events in a certain session (video + ads) should be the same.</td>
</tr>
<tr>
<td>1:asset:duration /</td>
<td>R</td>
<td>VideoInfo object</td>
<td>The video asset length of the main asset.</td>
</tr>
<tr>
<td>1:asset:length</td>
<td></td>
<td></td>
<td>(Renamed from length to duration in version 1.5)</td>
</tr>
<tr>
<td>s:asset:publisher</td>
<td>R</td>
<td>MediaHeartbeatConfig object</td>
<td>The publisher of the asset.</td>
</tr>
<tr>
<td>s:asset:video_id</td>
<td>R</td>
<td>VideoInfo object</td>
<td>An ID uniquely identifying the video in the publisher's catalog.</td>
</tr>
<tr>
<td>s:asset:type</td>
<td>R</td>
<td>Media SDK</td>
<td>The asset type (main or ad).</td>
</tr>
<tr>
<td>s:stream:type</td>
<td>R</td>
<td>VideoInfo object</td>
<td>The stream type. Can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• live</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• vod</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• linear</td>
</tr>
<tr>
<td>Name</td>
<td>Required / Optional</td>
<td>Data Source</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>s:user:id</td>
<td>O</td>
<td>Config object for mobile, app measurement VisitorID</td>
<td>User’s specifically set Visitor ID.</td>
</tr>
<tr>
<td>s:user:aid</td>
<td>O</td>
<td>Experience Cloud Org</td>
<td>The user’s analytics Visitor ID value.</td>
</tr>
<tr>
<td>s:user:mid</td>
<td>R</td>
<td>Experience Cloud Org</td>
<td>The user’s Experience cloud visitor ID value.</td>
</tr>
<tr>
<td>s:customer:customer_user_ids</td>
<td>O</td>
<td>MediaHeartbeatConfig object</td>
<td>All customer user IDs set on Audience Manager.</td>
</tr>
<tr>
<td>l:aam:loc_hint</td>
<td>R</td>
<td>MediaHeartbeatConfig object</td>
<td>AAM data sent on each payload after aa_start.</td>
</tr>
<tr>
<td>s:aam:blob</td>
<td>R</td>
<td>MediaHeartbeatConfig object</td>
<td>AAM data sent on each payload after aa_start.</td>
</tr>
<tr>
<td>s:sc:rsid</td>
<td>R</td>
<td>Report Suit ID (or IDs)</td>
<td>SiteCatalyst RSID where reports should be sent.</td>
</tr>
<tr>
<td>s:sc:tracking_server</td>
<td>R</td>
<td>MediaHeartbeatConfig object</td>
<td>SiteCatalyst tracking server.</td>
</tr>
<tr>
<td>h:sc:ssl</td>
<td>R</td>
<td>MediaHeartbeatConfig object</td>
<td>Whether the traffic is over HTTPS (if set to 1) or over HTTP (is set to 0).</td>
</tr>
<tr>
<td>s:sp:ovp</td>
<td>O</td>
<td>MediaHeartbeatConfig object</td>
<td>Set to &quot;primetime&quot; for Primetime players, or the actual OVP for other players.</td>
</tr>
<tr>
<td>s:sp:/sdk</td>
<td>R</td>
<td>MediaHeartbeatConfig object</td>
<td>The OVP version string.</td>
</tr>
<tr>
<td>s:sp:player_name</td>
<td>R</td>
<td>VideoInfo object</td>
<td>Video player name (the actual player software, used to identify the player).</td>
</tr>
<tr>
<td>s:sp:channel</td>
<td>O</td>
<td>MediaHeartbeatConfig object</td>
<td>The channel where the user is watching the content. For a mobile app, the app name. For a website, the domain name.</td>
</tr>
<tr>
<td>s:sp:hb_version</td>
<td>R</td>
<td>Media SDK</td>
<td>The version number of the VideoHeartbeat library issuing the call.</td>
</tr>
<tr>
<td>l:stream:bitrate</td>
<td>R</td>
<td>QoSInfo object</td>
<td>The current value of the stream bitrate (in bps).</td>
</tr>
<tr>
<td>Name</td>
<td>Required / Optional</td>
<td>Data Source</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Error Events</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s:event:source</td>
<td>R</td>
<td>Media SDK</td>
<td>The source of the error, either player-internal, or the application-level.</td>
</tr>
<tr>
<td>s:event:id</td>
<td>R</td>
<td>Media SDK</td>
<td>Error ID, uniquely identifies the error.</td>
</tr>
<tr>
<td><strong>Ad Events</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s:asset:ad_id</td>
<td>R</td>
<td>AdInfo object</td>
<td>The name of the ad.</td>
</tr>
<tr>
<td>s:asset:ad_sid</td>
<td>R</td>
<td>Media SDK</td>
<td>A unique identifier generated by the Media SDK, appended to all ad-related pings.</td>
</tr>
<tr>
<td>s:asset:pod_id</td>
<td>R</td>
<td>Media SDK</td>
<td>Pod ID inside the video. This value is computed automatically based on the following formula:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MD5(video_id) + &quot;_&quot; + [pod index]</td>
</tr>
<tr>
<td>s:asset:pod_position</td>
<td>R</td>
<td>AdBreakInfo object</td>
<td>Index of the ad inside the pod (the first ad has index 0, the second ad has index 1, etc.).</td>
</tr>
<tr>
<td>s:asset:resolver</td>
<td>R</td>
<td>AdBreakInfo object</td>
<td>The ad resolver.</td>
</tr>
<tr>
<td>s:meta:custom_ad_metadata.x</td>
<td>O</td>
<td>MediaHeartbeat object</td>
<td>The custom ad metadata.</td>
</tr>
<tr>
<td><strong>Chapter Events</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s:stream:chapter_sid</td>
<td>R</td>
<td>Media SDK</td>
<td>The unique identifier associated to the playback instance of the chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note</strong>: A chapter can be played multiple times due to seek-back operations performed by the user.</td>
</tr>
<tr>
<td>s:stream:chapter_name</td>
<td>O</td>
<td>ChapterInfo object</td>
<td>The chapter's friendly (i.e., human readable) name.</td>
</tr>
<tr>
<td>s:stream:chapter_id</td>
<td>R</td>
<td>Media SDK</td>
<td>The unique ID of the chapter. This value is computed automatically based on the following formula:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MD5(video_id) + &quot;_&quot; + chapter_pos</td>
</tr>
<tr>
<td>Name</td>
<td>Required / Optional</td>
<td>Data Source</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1:stream:chapter_pos</td>
<td>R</td>
<td>ChapterInfo object</td>
<td>The chapter's index in the list of chapters (starting with 1).</td>
</tr>
<tr>
<td>1:stream:chapter_offset</td>
<td>R</td>
<td>ChapterInfo object</td>
<td>The chapter's offset (expressed in seconds) inside the main content, excluding ads.</td>
</tr>
<tr>
<td>1:stream:chapter_length</td>
<td>R</td>
<td>ChapterInfo object</td>
<td>The chapter's duration (expressed in seconds).</td>
</tr>
<tr>
<td>s:meta:custom chapter_metadata.x</td>
<td>O</td>
<td>ChapterInfo object</td>
<td>Custom chapter metadata.</td>
</tr>
</tbody>
</table>

**Session End Event**

<table>
<thead>
<tr>
<th>Name</th>
<th>Required / Optional</th>
<th>Data Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s: event:type=end</td>
<td>R</td>
<td>Media SDK</td>
<td>The end event means that the SDK is sending a close call to the backend. On the receipt of this event, the backend will close the session for this video, and do no further processing.</td>
</tr>
</tbody>
</table>

**Debugging**

**SDK debugging**

You can enable or disable logging for Heartbeats.

The Video Heartbeat Library (VHL) 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 the VHL by setting the `debugLogging` flag on the Config object.

**Sample code for debug logging:**

- **Android** -

```java
// Media Heartbeat initialization
MediaHeartbeatConfig config = new MediaHeartbeatConfig();
config.debugLogging = true;

// Use this space for setting other config values
MediaHeartbeat _heartbeat = new MediaHeartbeat(this, config);
```

- **iOS** -

```objective-c
// Media Heartbeat Initialization
ADBMediaHeartbeatConfig *config = [[ADBMediaHeartbeatConfig alloc] init];
config.debugLogging = YES;

// Use this space for setting other config values
ADBMediaHeartbeat *_mediaHeartbeat =
    [[ADBMediaHeartbeat alloc] initWithDelegate:self config:config];
```
JavaScript -

```javascript
// Media Heartbeat initialization
var mediaConfig = new MediaHeartbeatConfig();
mediaConfig.debugLogging = true;
this._mediaHeartbeat = new MediaHeartbeat(mediaDelegate, mediaConfig, appMeasurement);
```

OTT (Chromecast, Roku) -

The ADBMobile library provides debug logging through the `setDebugLogging` method. Debug logging should be set to `false` for all the production apps.

Roku -

```javascript
ADBMobile().setDebugLogging(true)
```

Chromecast -

```javascript
ADBMobile.config.setDebugLogging(true)
```

**Using Adobe Bloodhound to Test Chromecast Applications** - During application development, Bloodhound allows you to view server calls locally, and optionally forward the data to Adobe collection servers. For more information about Bloodhound, see the following guides:

- Bloodhound 3.x for Mac
- Bloodhound 2.2 for Windows

⚠️ **Important**: As of April 30, 2017, Adobe Bloodhound has been sunset. Starting on May 1, 2017, no additional enhancements and no additional Engineering or Adobe Expert Care support will be provided.

Log messages follow this format:

`Format: [<timestamp>] [<level>] [tag] [message]`

Example: `16:10:29 GMT0700 (PDT).245 DEBUG plugin::player] Resolving qos.startTime: 0`

- **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*()` calls made by your application.

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

```javascript
[16:10:29 GMT0700 (PDT).222] [INFO] [plugin::player] #trackVideoLoad()
[16:10:29 GMT0700 (PDT).230] [INFO] [plugin::player] #trackSessionStart()
[16:10:29 GMT0700 (PDT).250] [INFO] [plugin::player] #trackPlay()
[16:10:29 GMT0700 (PDT).759] [INFO] [plugin::player] #trackChapterStart()
[16:10:44 GMT0700 (PDT).769] [INFO] [plugin::player] #trackAdStart()
[16:10:59 GMT0700 (PDT).752] [INFO] [plugin::player] #trackAdComplete()
[16:10:59 GMT0700 (PDT).770] [INFO] [plugin::player] #trackChapterStart()
[16:11:29 GMT0700 (PDT).734] [INFO] [plugin::player] #trackPause()
[16:11:29 GMT0700 (PDT).764] [INFO] [plugin::player] #trackComplete()
[16:11:29 GMT0700 (PDT).766] [INFO] [plugin::player] #trackVideoUnload()
Configure Adobe Debug

Accessing Adobe Debug
To access Adobe Debug:

1. Go to [www.marketing.adobe.com](http://www.marketing.adobe.com) and create a new Adobe Experience Cloud user.

   **Tip:** This login is not the same username/password you use to log into Adobe Analytics.

2. After you have an Experience Cloud account, contact your Adobe representative to request access to Adobe Debug.

3. After access has been granted, go to [https://debug.adobe.com](https://debug.adobe.com) and use Experience Cloud credentials to log in.

   ![Adobe Debug](image)

   The supported browsers for this tool are:

   - Google Chrome
   - Mozilla Firefox
   - Apple Safari
   - Microsoft Internet Explorer versions 9-11

   The recommended browsers are the latest versions of Chrome and Firefox.

Debug Proxy

Download
To download and configure the Debug Proxy:


   The supported operating systems are:

   - OS X 10.7 64-bit or higher
   - Windows 7.1 64-bit or higher
2. The Debug Proxy server will run on your local machine on port 33284 and will be set as the system proxy.
   You might need to adjust your browser setting based on the OS and the browser.

Download and install the SSL Certificate on desktop or apps

SSL Support

The first time you run Adobe Debug, a unique SSL certificate will be generated. If you support HTTPS traffic across desktop and/or Apps, you need to download and install our SSL certificate.

To download and install the SSL certificate:

1. After Adobe Debug has been installed and started, go to http://proxy.debug.adobe.com/ssl and download the certification.
2. Import the certificate.

   **Mac OS**
   1. Double-click the root CA certificate to open it in Keychain Access.
   2. The root CA certificate appears in login.
   3. Move (drag) the root CA certificate to System.
   4. You must copy the certificate to System to ensure that it is trusted by all users and local system processes.
   5. Open the root CA certificate, expand Trust, select Always Trust, and save your changes.

   **Windows**
   1. Complete one of the following procedures:
      - Adding certificates to the Trusted Root Certification Authorities store for a local computer
      - How To Import a Trusted Root Certification Authority In Windows 7/Vista/XP
      You might need to quit and reopen your browser to see the change.
   2. For Firefox, complete the procedure in Installing root certificate in Mozilla Firefox.
      You might need to quit and reopen Firefox to see the change.

   **iOS devices**
   1. Set your iOS device to use Adobe Debug as its HTTP proxy by clicking **Settings app > Wifi settings**.
Safari will prompt you to install the SSL certificate.

Install the SSL certificate for your mobile device

If you are missing the HTTPS calls in Adobe Debug, you must install the SSL Certificate for Adobe Debug on the mobile device.

iOS

To install the SSL certificate on an iOS device:

1. On your laptop, turn on the Debug Proxy, and go to Adobe Debug.
2. Complete the following steps on your iOS device:
   a. Turn your device to airplane mode.
   b. Select the same Wi-Fi signal that is used by your laptop.
   c. On your laptop, manually set the IP and port shown on the Debug Proxy app.
   d. Open an Apple Safari browser window.
   e. Go to http://proxy.debug.adobe.com/ssl.
   f. Download and install the SSL certificate.
3. On your laptop, start your Adobe Debug session.
4. Start testing on your iOS device.

Android

To install the SSL certificate on an Android device:

1. On your laptop turn on the Debug Proxy and go to Adobe Debug.
2. Complete the following steps on your Android device:
   a. Set your device to Airplane Mode.
   b. Select the same Wi-Fi signal that is used by your laptop.
   c. On your laptop, manually set the IP and port shown on the Debug Proxy app.
   d. Open a browser window.
   e. Go to http://proxy.debug.adobe.com/ssl.
   f. Download and install the SSL certificate.
3. On your laptop, start your Adobe Debug session.
4. Start testing on your Android device.

Create a new Debug report

To create a new Debug report:

1. In Create New Debug Report, select the following:
2. Complete the fields with the following information:

- **Name the Report** - Enter the player name and date so that you can easily track the player during certification and keep brands and platforms separate.

- **Adobe Analytics**
  - **User Name** and **Shared Secret** - These fields are optional, but you can add your web services API credentials to Adobe Debug to display the variable names and variable settings for the report suite.

You can access in one of the following ways:

- **Analytics > Admin > Company Settings > Web Services**
- **Analytics > Admin > User Management > Users > Individual User Settings**

To create a web services API credential for a new user, in **User Management**, add the user to the **Web Service Access** user group.
• **Default Endpoint** – The data in this field is provided by Adobe and cannot be changed.
• **Extra Endpoint** – Add CNAMEs, if you use them, for tracking server like metrics.companyname.com

**Video Heartbeats**

• **Default Endpoint** – The data in this field is provided by Adobe and cannot be changed.
• **Extra Endpoint** – Add CNAMEs, if you use them, for tracking server like metrics.companyname.com.

**Debug dashboards and reports**

Adobe Debug provides reporting, in real-time, so you can view hits and metadata that are being sent during video playback. Each of these reports can be saved within Debug.

For certification, these reports can be shared by simply copying the URL and sending the link, e.g., within a ZenDesk ticket.

💡 **Tip:** Only one session can be active at a time. The active session can be opened from the Dashboard.

![Adobe Debug dashboard](image-url)

**Analytics in OTT apps**

**Track app states**

States are the different screens or views in your application. Each time a new state is displayed in your application, for example, when a user navigates from the home page to the video details screen, you should send a `trackState` call.

`trackState` is typically called each time a new screen is loaded.

• **Roku** -
  
  ADBMobile().trackState("State Name", {})

• **Chromecast** -
  
  ADBMobile.analytics.trackState("State Name", {});
The state name is reported in the View State variable in Adobe Mobile services, and a view is recorded for each trackState call. In other Analytics interfaces, View State is reported as Page Name and state views is reported as page views.

In addition to the State Name, you can send additional context data with each track state call:

- **Roku**
  ```javascript
dictionary = { }
dictionary["myapp.login.LoginStatus"] = "logged in"
ADBMobile().trackState("Home Screen", dictionary)
```

- **Chromecast**
  ```javascript
  var dictionary = { ];
dictionary["myapp.login.LoginStatus"] = "logged in";
ADBMobile.analytics.trackState("Home Screen", dictionary);
```

💡 **Note:** Context data values must be mapped to custom variables in Adobe Mobile services.

States are typically viewed by using a pathing report, so you can see how users navigate your app and which states are most commonly viewed.

### Track app actions

Actions are the events that occur in your app that you want to measure.

Each action has one or more corresponding metrics that are incremented each time the event occurs. For example, you might send a trackAction call for each new subscription, each time a content is rated, or each time a level is completed. Actions are not tracked automatically, so call trackAction when an event that you want to track occurs, and map the action to a custom event.

1. When an event you want to track occurs, call trackAction.
   ```javascript
   ADBMobile().trackAction("myapp.ActionName", {})
   ADBMobile.analytics.trackAction("myapp.ActionName", {});
   ```

2. Map the action to a custom event.
   ```javascript
   dictionary = {}
dictionary["myapp.social.SocialSource"] = "Twitter"
ADBMobile().trackAction("myapp.SocialShare", dictionary)
   
   var dictionary = { ];
dictionary["myapp.social.SocialSource"] = "Twitter";
ADBMobile.analytics.trackAction("myapp.SocialShare", dictionary);
   ```

You can also send additional context data with each track action call.

### Set user IDs

The user ID is a unique custom visitor identifier defined by the application for a user.

Set and get the unique user ID on the ADBMobile SDK as follows:

1. **Set:**
• Roku:
  `ADBMobile().setUserIdentifer("app-generated-unique-id")`

• Chromecast:
  `ADBMobile().config.setUserIdentifer("app-generated-unique-id");`

2. Get:

• Roku:
  `vid = ADBMobile().userIdentifer();`

• Chromecast:
  `vid = ADBMobile().config.getUserIdentifer();`

OTT and Audience Manager

For details on integrating OTT apps and Audience Manager, see *Audience Manager enablement*.

OTT and Experience Cloud

For details on integrating OTT apps with Experience Cloud, see the "Set Up" topics for Chromecast and Roku:

• *Set up Chromecast*
• *Set Up Roku*

Cookbook

Handling application interrupts during playback

Playback in a video application can be interrupted in a variety of ways: a user explicitly presses pause, or when a user puts the application into the background. Regardless of what causes an interruption in video playback, the tracking instructions are the same:

1. Call `trackPause` when the application is interrupted (goes to background, video pauses, etc.).
2. Call `trackPlay` when the application returns to the foreground and/or the video resumes playing.

💡 *Note:* The Media Analytics team has seen instances where customers called `trackSessionStart` when their app returned from the background. Doing this results in the playback up to that point not counting towards the total playback time, along with losing earlier progress markers, segments, and so on. Instead, call `trackPlay` when the app returns and/or the video resumes playing.

FAQ about handling application interrupts:

• *How long should an app be backgrounded before the session closes?*

If the application allows background playback, it can continue tracking by calling our APIs and we will send all our regular tracking pings. I have not seen any video app allow background playback except youtube red, however, all audio apps allow this. If the application does not allow background playback, then it is advisable to stay in pause...
state for 1 minute and then end the tracking session. The application can not continue sending pause pings as in most cases it cannot determine if the user is going to return back to continue viewing the video or when it is going to be killed. It is also a bad experience to keep sending pings when in the background.

• **What is the correct way to handle re-starting tracking after the app has been in the background for a long time?**

  The application should call `trackSessionEnd` to end the tracking session. From Version 2.1, the SDK sends an "end" ping to notify the back-end that the tracking session is closed.

• **What about restarting the same session?**

  For detailed instructions on restarting a tracking session, see this page: *Manually resume a previously closed session*. The SDK sends a resume ping to notify the back-end that the user is manually resuming the session.

### Resolving main:play appearing between ads

**PROBLEM**

In some ad tracking scenarios, you could encounter `main:play` calls occurring unexpectedly between the end of one ad and the start of the next ad. If the delay between the ad complete call and the next ad start call is greater than 250 milliseconds, the Media Analytics (MA) SDK will fall back to sending `main:play` calls. If this fallback to `main:play` occurs during a pre-roll ad break, the content start metric may be set early.

A gap between ads such as described above is interpreted by the Media SDK as main content, because there is no overlap there with any ad content. The Media SDK does not have any ad information set on it, and the player is in the playing state. If there is no Ad information, and the player state is playing, then the Media SDK credits the duration of the gap towards main content by default. It cannot credit playback duration toward null ad information.

**IDENTIFICATION**

While using Adobe Debug or a network packet sniffer such as Charles, if you see the following heartbeat calls in this order during a pre-roll ad break:

- **Session Start:** `s:event:type=start & s:asset:type=main`
- **Ad Start:** `s:event:type=start & s:asset:type=ad`
- **Ad Play:** `s:event:type=play & s:asset:type=ad`
- **Ad Complete:** `s:event:type=complete & s:asset:type=ad`
- **Main Content Play:** `s:event:type=play & s:asset:type=main` *(unexpected)*
- **Ad Start:** `s:event:type=start & s:asset:type=ad`
- **Ad Play:** `s:event:type=play & s:asset:type=ad`
- **Ad Complete:** `s:event:type=complete & s:asset:type=ad`
- **Main Content Play:** `s:event:type=play & s:asset:type=main` *(expected)*

**RESOLUTION**

*Delay triggering the Ad Complete call.*

Handle the gap from within the player by calling `trackEvent:AdComplete` late for the first ad, followed immediately by `trackEvent:AdStart` for the second ad. The app should hold off on calling the AdComplete event after the first ad finishes. Make sure you call `trackEvent:AdComplete` for the last ad in the ad break. If the player can identify that the current ad asset is the final one in the ad break, call `trackEvent:AdComplete` immediately. This resolution will result in less than 1 second of additional ad time spent being attributed to the preceding ad unit.
On ad break start, including pre-roll:

• Create the adBreak object instance for the ad break; for example, adBreakObject.
• Call trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakObject);

On every ad asset start:

• Call trackEvent(MediaHeartbeat.Event.AdComplete); (New!)

  Note: Call this only if the previous ad wasn’t complete. Consider a Boolean value to maintain an “isinAd” state for the previous ad.

• Create the ad object instance for the ad asset: for example, adObject.
• Populate the ad metadata, adCustomMetadata.
• Call trackEvent(MediaHeartbeat.Event.AdStart, adObject, adCustomMetadata);
• Call trackPlay() if this is the first ad in a pre-roll ad break.

On every ad asset complete:

• Do not make a call (New!)

  Note: If the application knows it is the last ad in the ad break, call trackEvent:AdComplete here and skip setting trackEvent:AdComplete in the trackEvent:AdBreakComplete

On ad skip:

• Call trackEvent(MediaHeartbeat.Event.AdSkip);

On ad break complete:

• Call trackEvent(MediaHeartbeat.Event.AdComplete); (New!)

  Note: If this step is already performed above as part of the last trackEvent:AdComplete call then this can be skipped.

• Call trackEvent(MediaHeartbeat.Event.AdBreakComplete);

Resuming inactive sessions

Long Pauses -

The VA library automatically tracks how long the video playback is in one of the following inactive states:

• Paused
• Seeking
• Stalled
• Buffering

If a video tracking session remains in an inactive state for longer than 30 minutes, the session will automatically be closed. If the user resumes after a previously inactive video tracking session (trackPlay), Media Heartbeat automatically creates a new video session using the previously used video information and metadata, and sends a resume heartbeat event. For more information, see Audio and video parameters.
Manually resume previously closed session -

The VA library will only automatically resume sessions if the application was not closed. If the application stores user data and has the capability to resume a previously closed video, it is possible to manually trigger a resume event. When starting the video tracking session, set the optional Video Resumed property.

**Android:**

```java
public void onVideoLoad(Observable observable, Object data) {
    // Set MediaHeartbeat.MediaObjectKey.VideoResumed to true
    // Replace <MEDIA_NAME> with the video name.
    // Replace <MEDIA_ID> with a video unique identifier.
    // Replace <MEDIA_LENGTH> with the video length.
    MediaObject mediaInfo = MediaHeartbeat.createMediaObject(  
        <MEDIA_NAME>,
        <MEDIA_ID>,
        <MEDIA_LENGTH>,
        MediaHeartbeat.StreamType.VOD
    );

    // Set to true if this is a resume playback scenario
    mediaInfo.setValue(MediaHeartbeat.MediaObjectKey.VideoResumed, true);
    _heartbeat.trackSessionStart(mediaInfo, videoMetadata);
}
```

**iOS:**

```objective-c
- (void)onMainVideoLoaded:(NSNotification *)notification {
    // Replace <MEDIA_NAME> with the video name.
    // Replace <MEDIA_ID> with a video unique identifier.
    // Replace <MEDIA_LENGTH> with the video length.
    ADBMediaObject *mediaObject =  
        [ADBMediaHeartbeat createMediaObjectWithName:<MEDIA_NAME>  
            mediaId:<MEDIA_ID>  
            length:<MEDIA_LENGTH>  
            streamType:ADBMediaHeartbeatStreamTypeVOD];

    // Set to YES if this user is resuming a previously closed media session
    [mediaObject setValue:@(YES) forKey:ADBMediaObjectKeyVideoResumed];
    [_mediaHeartbeat trackSessionStart:mediaObject data:videoMetadata];
}
```

**JavaScript:**

```javascript
_onVideoLoad = function () {
    // Replace <MEDIA_NAME> with the video name.
    // Replace <MEDIA_ID> with a video unique identifier.
    // Replace <MEDIA_LENGTH> with the video length.
    var mediaObject =  
        MediaHeartbeat.createMediaObject(<MEDIA_NAME>,  
            <MEDIA_ID>,  
            <MEDIA_LENGTH>,  
            MediaHeartbeat.StreamType.VOD);

    // Set to true if this user is resuming a previously closed media session
    mediaObject.setValue(MediaObjectKey.VideoResumed, true);
    this._mediaHeartbeat.trackSessionStart(mediaObject, contextData);
};
```
Tracking in SceneGraph (Roku)

Introduction

Roku has introduced a new programming framework for developing applications: the SceneGraph XML programming framework. This new framework features two new key concepts:

- SceneGraph rendering of the application screens
- XML configuration of the SceneGraph screens

The Adobe Mobile SDK for Roku is written in BrightScript. The SDK uses many components that are not available for an app running on SceneGraph (for example, threads). Therefore, a Roku app developer intending to use the SceneGraph framework cannot call Adobe Mobile SDK APIs (the latter are similar to those available in legacy BrightScript apps).

Architecture

To add SceneGraph support to the AdobeMobile SDK, Adobe has added a new API that creates a connector bridge between the AdobeMobile SDK and adbmobileTask. The latter is a SceneGraph node used for the SDK’s API execution. (Usage of adbmobileTask is explained in detail throughout the rest of this document.)

The connector bridge is designed to perform as follows:

- The bridge returns a SceneGraph-compatible instance of the AdobeMobile SDK. The SceneGraph-compatible SDK has all of the APIs that the legacy SDK exposes.
- You use the AdobeMobile SDK APIs in SceneGraph in a very similar way to how you used the legacy APIs.
- The bridge also exposes a mechanism to listen for callbacks for APIs that return some data.
Components

SceneGraph Application:

- Consumes AdobeMobileLibrary APIs via the SceneGraph connector bridge APIs.
- Registers for response callbacks on adbmobileTask for expected output data variables.

AdobeMobileLibrary:

- Exposes a set of public APIs (Legacy), including the connector bridge API.
- Returns a SceneGraph connector instance that wraps all legacy public APIs.
- Communicates with an adbmobileTask SceneGraph node for execution of APIs.

adbmobileTask Node:

- A SceneGraph task node that executes AdobeMobileLibrary APIs on a background thread.
- Serves as a delegate to return data back to application scenes.
Public SceneGraph APIs

Table 43: adbmobile.brs

<table>
<thead>
<tr>
<th>Name</th>
<th>API Signature</th>
<th>Input</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>getADBMobileConnectorInstance</td>
<td>ADBMobile().getADBMobileConnectorInstance()</td>
<td>adbmobileTask</td>
<td>ADBMobileConnector</td>
</tr>
</tbody>
</table>

*Comments: Refer to the ADBMobileConnector API reference for details.*

<table>
<thead>
<tr>
<th>Name</th>
<th>API Signature</th>
<th>Input</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sgConstants</td>
<td>ADBMobile().sgConstants()</td>
<td>None</td>
<td>SceneGraphConstants</td>
</tr>
</tbody>
</table>

*Comments: Refer to the SceneGraphConstants API reference for details.*

Table 44: adbmobileTask Node

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Default</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>adbmobileApiCall</td>
<td>assocarray</td>
<td>Invalid</td>
<td><strong>Do NOT</strong> modify this field or let it be used by the Application. This field is used by the ADBMobile SceneGraphConnector to route API calls via SceneGraph nodes and to fetch responses. Therefore, this key/field is reserved for AdobeMobileSDK for SceneGraph compatibility. <strong>Important:</strong> Any modifications to this field may result in AdobeMobileSDK functioning incorrectly.</td>
</tr>
</tbody>
</table>
| adbmobileApiResponse    | assocarray   | Invalid | **Read-Only**

All of the APIs executed on AdobeMobileSDK will return responses on this field. Register for a callback to listen for updates to this field in order to receive response objects. Following is the format for the response object:

```javascript
response = {
    "apiName" : <SceneGraphConstants.API_NAME>
    "returnValue" : <API_RESPONSE>
}
```

An instance of this response object will be sent for any API call on AdobeMobileSDK that is expected to return a value as per the API reference guide. For example, an API call for `visitorMarketingCloudID()` will return the following response object:

```javascript
response = {
    "apiName" : m.adbmobileConstants.
    VISITOR_MARKETING_CLOUD_ID
    "returnValue" : 
    "07050xxxx25671xxxx33760xxxx72644xxxx14"
}
```

OR, response data can be invalid as well:

```javascript
response = {
    "apiName" : m.adbmobileConstants.
    "apiName" : "Invalid"
}
```
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Default</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VISITOR_MARKETING_CLOUD_ID</td>
<td></td>
<td>&quot;returnValue : invalid}</td>
<td></td>
</tr>
</tbody>
</table>

Table 45: SceneGraphConstants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API_RESPONSE</td>
<td>Used to retrieve the response object from adbmobileTask node's adbmobileApiResponse field</td>
</tr>
<tr>
<td>DEBUG_LOGGING</td>
<td>Used as apiName for getDebugLogging</td>
</tr>
<tr>
<td>PRIVACY_STATUS</td>
<td>Used as apiName for getPrivacyStatus</td>
</tr>
<tr>
<td>TRACKING_IDENTIFIER</td>
<td>Used as apiName for trackingIdentifier</td>
</tr>
<tr>
<td>USER_IDENTIFIER</td>
<td>Used as apiName for userIdentifier</td>
</tr>
<tr>
<td>VISITOR_MARKETING_CLOUD_ID</td>
<td>Used as apiName for visitorMarketingCloudID</td>
</tr>
<tr>
<td>AUDIENCE_VISITOR_PROFILE</td>
<td>Used as apiName for audienceVisitorProfile</td>
</tr>
<tr>
<td>AUDIENCE_DPID</td>
<td>Used as apiName for audienceDpid</td>
</tr>
<tr>
<td>AUDIENCE_DPUUID</td>
<td>Used as apiName for audienceDpuuid</td>
</tr>
</tbody>
</table>

Table 46: ADBMobileConnector

<table>
<thead>
<tr>
<th>Category</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constants</td>
<td>sceneGraphConstants</td>
<td>Returns an object containing SceneGraphConstants. Refer to the table above for details.</td>
</tr>
<tr>
<td>Debug Logging</td>
<td>setDebugLogging</td>
<td>SceneGraph API to set debug logging on the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td>getDebugLogging</td>
<td>SceneGraph API to get debug logging from the ADBMobile SDK.</td>
</tr>
<tr>
<td>Privacy Status / Opt-Out</td>
<td>setPrivacyStatus</td>
<td>SceneGraph API to set privacy status on the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td>getPrivacyStatus</td>
<td>SceneGraph API to get privacy status from the ADBMobile SDK.</td>
</tr>
</tbody>
</table>

*For more information refer to the Debug Logging section of the legacy SDK.*

*For more information, refer to the Opt-Out/Privacy Status section of the legacy SDK.*
<table>
<thead>
<tr>
<th>Category</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analytics</strong></td>
<td><strong>trackState</strong></td>
<td>SceneGraph API to track state on the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>trackAction</strong></td>
<td>SceneGraph API to track action on the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>trackingIdentifier</strong></td>
<td>SceneGraph API to get a tracking identifier from the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>userIdentifier</strong></td>
<td>SceneGraph API to get a user identifier from the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>setUserIdentifier</strong></td>
<td>SceneGraph API to set the user identifier on the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>getAllIdentifiers</strong></td>
<td>SceneGraph API retrieves all user identities known and persisted by the Roku SDK.</td>
</tr>
<tr>
<td><strong>Experience Cloud</strong></td>
<td><strong>visitorSyncIdentifiers</strong></td>
<td>SceneGraph API to sync Experience Cloud identifiers on the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>visitorMarketingCloudID</strong></td>
<td>SceneGraph API to get Visitor Experience Cloud ID from the ADBMobile SDK.</td>
</tr>
<tr>
<td><strong>Audience Manager</strong></td>
<td><strong>audienceSubmitSignal</strong></td>
<td>SceneGraph API to send an audience management signal with trait.</td>
</tr>
<tr>
<td></td>
<td><strong>audienceVisitorProfile</strong></td>
<td>SceneGraph API to get an audience manager visitor profile from the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>audienceDpid</strong></td>
<td>SceneGraph API to get an audience Dpid from the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>audienceDpuuid</strong></td>
<td>SceneGraph API to get an audience Dpuuid from the ADBMobile SDK.</td>
</tr>
<tr>
<td></td>
<td><strong>audienceSetDpidAndDpuuid</strong></td>
<td>SceneGraph API to set audience Dpid and Dpuuid on the ADBMobile SDK.</td>
</tr>
<tr>
<td><strong>MediaHeartbeat</strong></td>
<td><strong>mediaTrackLoad</strong></td>
<td>SceneGraph API to load video content for MediaHeartbeat tracking.</td>
</tr>
<tr>
<td></td>
<td><strong>mediaTrackStart</strong></td>
<td>SceneGraph API to start video tracking session using MediaHeartbeat.</td>
</tr>
<tr>
<td></td>
<td><strong>mediaTrackUnload</strong></td>
<td>SceneGraph API to unload video content from MediaHeartbeat tracking.</td>
</tr>
<tr>
<td></td>
<td><strong>mediaTrackPlay</strong></td>
<td>SceneGraph API to track playback of video content.</td>
</tr>
<tr>
<td>Category</td>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>mediaTrackPause</td>
<td>SceneGraph API to track paused video content.</td>
</tr>
<tr>
<td></td>
<td>mediaTrackComplete</td>
<td>SceneGraph API to track playback complete for video content.</td>
</tr>
<tr>
<td></td>
<td>mediaTrackError</td>
<td>SceneGraph API to track playback errors.</td>
</tr>
<tr>
<td></td>
<td>mediaTrackEvent</td>
<td>SceneGraph API to track playback events during tracking. For example: Ads, Chapters.</td>
</tr>
<tr>
<td></td>
<td>mediaUpdatePlayhead</td>
<td>SceneGraph API to send playhead updates to MediaHeartbeat during video tracking.</td>
</tr>
<tr>
<td></td>
<td>mediaUpdateQoS</td>
<td>SceneGraph API to send QoS updates to MediaHeartbeat during video tracking.</td>
</tr>
</tbody>
</table>

In addition to methods above, ADBMobileConnector also exposes all the ADBMobile constants for the following:

**Table 47: ADBMobile Constants**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versioning</td>
<td>version</td>
<td>Constant for retrieving AdobeMobileLibrary version info</td>
</tr>
<tr>
<td>Privacy/opt-out</td>
<td>PRIVACY_STATUS_OPT_IN</td>
<td>Constant for privacy status opted in</td>
</tr>
<tr>
<td></td>
<td>PRIVACY_STATUS_OPT_OUT</td>
<td>Constant for privacy status opted out</td>
</tr>
<tr>
<td>MediaHeartbeat Constants</td>
<td>Refer to the constants on this page: Media Heartbeat Methods.</td>
<td>Use these constants for using MediaHeartbeat APIs</td>
</tr>
<tr>
<td>Standard Metadata</td>
<td>Refer to the constants on this page: Standard Metadata Parameters.</td>
<td>Use these constants to attach Standard Video/Ad metadata in MediaHeartbeat APIs</td>
</tr>
</tbody>
</table>

Globally defined utility MediaHeartbeat APIs on the legacy AdobeMobileLibrary are accessible as is in the SceneGraph environment because they do not use any components for Brightscript that are unavailable in SceneGraph nodes. For more information on these methods, refer to the table below:

**Table 48: Global Methods for MediaHeartbeat**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adb_media_init_medainfo</td>
<td>This method returns an initialized Media Information object</td>
</tr>
<tr>
<td></td>
<td>Function adb_media_init_medainfo(name As String, id As String, length As Double, streamType As String) As Object</td>
</tr>
<tr>
<td>adb_media_init_adinfo</td>
<td>This method returns initialized Ad Information object</td>
</tr>
<tr>
<td></td>
<td>Function adb_media_init_adinfo(name As String, id As String, position As Double, length As Double) As Object</td>
</tr>
</tbody>
</table>
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adb_media_init_chapterinfo</code></td>
<td>This method returns initialized Chapter Information object.</td>
</tr>
<tr>
<td><code>adb_media_init_adbreakinfo</code></td>
<td>Function <code>adb_media_init_adbreakinfo(name As String, startTime as Double, position as Double) As Object</code></td>
</tr>
<tr>
<td><code>adb_media_init_qosinfo</code></td>
<td>This method returns initialized QoS Information object.</td>
</tr>
<tr>
<td><code>adb_media_init_chapterinfo</code></td>
<td>Function <code>adb_media_init_chapterinfo(name As String, position As Double, length As Double, startTime As Double) As Object</code></td>
</tr>
</tbody>
</table>

### Implementation

1. **Download the Roku Library**
   

2. **Set Up Your Development Environment**
   
   a. Copy `adbmobile.brs` (AdobeMobileLibrary) into your pkg:/source/ directory.
   b. For Scene Graph support, copy `adbmobileTask.brs` and `adbMobileTask.xml` into your pkg:/components/ directory.

3. **Initialize**
   
   a. Import `adbmobile.brs` into your Scene.
      
      ```xml
      <script type="text/brightscript" uri="pkg:/source/adbmobile.brs" />
      ```
   b. Create an instance of `adbmobileTask` node into your Scene.
      
      ```javascript
      m.adbmobileTask = createObject("roSGNode", "adbmobileTask")
      ```
   c. Get an instance of `adbmobile` connector for SceneGraph using the `adbmobileTask` instance.
      
      ```javascript
      m.adbmobile = ADBMobile().getADBMobileConnectorInstance(m.adbmobileTask)
      ```
   d. Get `adbmobile` SG constants.
      
      ```javascript
      m.adbmobileConstants = m.adbmobile.sceneGraphConstants()
      ```
   e. Register a callback for receiving response object for all AdbMobile API calls.
      
      ```javascript
      m.adbmobileTask.ObserveField(m.adbmobileConstants.API_RESPONSE, "onAdbmobileApiResponse")
      ` Sample implementation of the callback
      ' Listen for all the constants for which API calls are made on the SDK
      function onAdbmobileApiResponse() as void
      responseObject = m.adbmobileTask[m.adbmobileConstants.API_RESPONSE]
      if responseObject <> invalid
        methodName = responseObject.apiName
        retVal = responseObject.returnValue
      ```
if methodName = m.adbmobileConstants.DEBUG_LOGGING
  if retVal
    print "API Response: DEBUG LOGGING: " + "True"
  else
    print "API Response: DEBUG LOGGING: " + "False"
  endif
else if methodName = m.adbmobileConstants.PRIVACY_STATUS
  print "API Response: PRIVACY STATUS: " + retVal
else if methodName = m.adbmobileConstants.TRACKING_IDENTIFIER
  if retVal <> invalid
    print "API Response: TRACKING IDENTIFIER: " + retVal
  else
    print "API Response: TRACKING IDENTIFIER: " + "invalid"
  endif
else if methodName = m.adbmobileConstants.USER_IDENTIFIER
  if retVal <> invalid
    print "API Response: USER IDENTIFIER: " + retVal
  else
    print "API Response: USER IDENTIFIER: " + "invalid"
  endif
else if methodName = m.adbmobileConstants.VISITOR_MARKETING_CLOUD_ID
  if retVal <> invalid
    print "API Response: MCID: " + retVal
  else
    print "API Response: MCID: " + "invalid"
  endif
else if methodName = m.adbmobileConstants.AUDIENCE_DPID
  if retVal <> invalid
    print "API Response: AUDIENCE DPID: " + retVal
  else
    print "API Response: AUDIENCE DPID: " + "invalid"
  endif
else if methodName = m.adbmobileConstants.AUDIENCE_DPUUID
  if retVal <> invalid
    print "API Response: AUDIENCE DPUUID: " + retVal
  else
    print "API Response: AUDIENCE DPUUID: " + "invalid"
  endif
else if methodName = m.adbmobileConstants.AUDIENCE_VISITOR_PROFILE
  if retVal <> invalid
    print "API Response: AUDIENCE VISITOR PROFILE: Valid Object"
  else
    print "API Response: AUDIENCE VISITOR PROFILE: " + "invalid"
  endif
endif
endif
end function

Sample Implementation

Sample API calls on Legacy SDK

'get an instance of SDK
m.adbmobile = ADBMobile()

'execute setter APIs
m.adbmobile.setDebugLogging(true)

'execute getter APIs
defaultLogging = m.adbmobile.getDebugLogging()

Sample API calls on SG SDK

'create adbmobileTask instance
m.adbmobileTask = createObject("roSGNode", "adbmobileTask")
'get an instance of SDK using task instance
m.adbmobile = ADBMobile().getADBMobileConnectorInstance(m.adbmobileTask)
m.adbmobileConstants = m.adbmobile.sceneGraphConstants()
'execute setter APIs
m.adbmobile.setDebugLogging(true)

'execute getter APIs
m.adbmobileTask.ObserverField(m.adbConstants.API_RESPONSE,
   "onAdbmobileApiResponse")
m.adbmobile.getDebugLogging()

'listen for return data in registered callbacks
function onAdbmobileApiResponse() as void
   responseObject = m.adbmobileTask[m.adbmobileConstants.API_RESPONSE]
   if responseObject <> invalid
      methodName = responseObject.apiName
      retVal = responseObject.returnValue
      if methodName = m.adbmobileConstants.DEBUG_LOGGING
         if retVal
            print "API Response: DEBUG LOGGING: " + "True"
         else
            print "API Response: DEBUG LOGGING: " + "False"
      endif
   endif
end function

Media Analytics 1.x to 2.x migration

Migration overview

The migration from VHL 1.x to VHL 2.x is straightforward, with the new version featuring simplified APIs for initialization, configuration, and player delegates.

Here are the primary differences between 1.x and 2.x:

- **Plugins, Delegates** - You no longer need to implement plugins and delegates for Analytics, VideoPlayer, and Heartbeat.

- **Configuration** - You no longer need to instantiate configurations for the 1.x plugins.

In versions 2.x, all of the public methods are consolidated into the `MediaHeartbeat` class to make implementation easier on developers. Also, all configs are now consolidated into the `MediaHeartbeatConfig` class. These new APIs are described in detail here: [1.x to 2.x API conversion](#).

In version 2.x, you no longer need to instantiate configs for the Analytics, VideoPlayer, and Heartbeat plugins. In the 2.x SDK you only need to instantiate the `MediaHeartbeat` class with `MediaHeartbeatDelegate` and `MediaHeartbeatConfig` instances. This is the only implementation that is required to initialize Media Analytics.

With the initialization of `MediaHeartbeat`, you can safely delete all of the implementation for Analytics Plugin, VideoPlayer Plugin and Heartbeat Plugin. Also, remove all the existing implementation for VideoHeartbeat initialization that takes in an array of plugins as an input. You can see side-by-side comparisons of the 1.x and 2.x implementations here: [Code comparison: 1.x to 2.x](#)
**Code comparison: 1.x to 2.x**

All of the configuration parameters and tracking APIs are now consolidated into the `MediaHeartbeats` and `MediaHeartbeatConfig` classes.

**Configuration API changes:**

- AdobeHeartbeatPluginConfig.sdk - Renamed to `MediaConfig.appVersion`
- MediaHeartbeatConfig.playerName - Now set through `MediaHeartbeatConfig` instead of `VideoPlayerPluginDelegate`
- (For JavaScript only): The `AppMeasurement` instance - Now sent through the `MediaHeartbeat` constructor.

**Configuration properties changes:**

- sdk - Renamed to `appVersion`
- publisher - Removed; Experience Cloud Org ID is used instead as a publisher
- quiteMode - Removed

The following tables provide side-by-side code comparisons between 1.x and 2.x, covering Initialization, Core Playback, Ad Playback, Chapter Playback, and some additional events.

**Table 49: VHL Code Comparison: INITIALIZATION**

<table>
<thead>
<tr>
<th>1.x API</th>
<th>2.x API</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object Initialization</strong></td>
<td><strong>...</strong></td>
</tr>
<tr>
<td>1.x:</td>
<td>2.x:</td>
</tr>
<tr>
<td>• Heartbeat()</td>
<td>• MediaHeartbeat()</td>
</tr>
<tr>
<td>• VideoPlayerPlugin()</td>
<td>• MediaHeartbeatConfig()</td>
</tr>
<tr>
<td>• AdobeAnalyticsPlugin()</td>
<td></td>
</tr>
<tr>
<td>• HeartbeatPlugin()</td>
<td></td>
</tr>
</tbody>
</table>

**Set up the video player plugin:**

```javascript
var mediaConfig = new MediaHeartbeatConfig();
mediaConfig.trackingServer = Configuration.HEARTBEAT.TRACKING_SERVER;
mediaConfig.playerName = Configuration.PLAYER.NAME;
mediaConfig.debugLogging = true;
mediaConfig.channel = Configuration.HEARTBEAT.CHANNEL;
mediaConfig.ssl = false;
mediaConfig.ovp = Configuration.HEARTBEAT.OVP;
mediaConfig.appVersion = Configuration.HEARTBEAT.SDK;
this._mediaHeartbeat = new MediaHeartbeat(
  new SampleMediaHeartbeatDelegate(this._player),
  mediaConfig,
  appMeasurement);
```

**Media Heartbeat initialization:**

```javascript
this._playerPlugin = new VideoPlayerPlugin(
  new SampleVideoPlayerPluginDelegate(this._player));
var playerPluginConfig = new VideoPlayerPluginConfig();
playerPluginConfig.debugLogging = true;
// Set up the AppMeasurement plugin
this._aaPlugin = new AdobeAnalyticsPlugin(
  appMeasurement,
  new SampleAdobeAnalyticsPluginDelegate());
var aaPluginConfig = new AdobeAnalyticsPluginConfig();
aaPluginConfig.channel = Configuration.HEARTBEAT.CHANNEL;
aaPluginConfig.debugLogging = true;
this._aaPlugin.configure(aaPluginConfig);
```
```javascript
// Set up the AdobeHeartbeat plugin
var ahPlugin = new AdobeHeartbeatPlugin(new SampleAdobeHeartbeatPluginDelegate());
var ahPluginConfig = new AdobeHeartbeatPluginConfig(
    configuration.HEARTBEAT.TRACKING_SERVER,
    configuration.HEARTBEAT.PUBLISHER);
ahPluginConfig.ovp = configuration.HEARTBEAT.OVP;
ahPluginConfig.sdk = configuration.HEARTBEAT.SDK;
ahPluginConfig.debugLogging = true;
ahPlugin.configure(ahPluginConfig);

var plugins = [this._playerPlugin, this._aaPlugin, ahPlugin];

// Set up and configure the heartbeat library
this._heartbeat = new Heartbeat(new SampleHeartbeatDelegate(), plugins);
var configData = new HeartbeatConfig();
cfgData.debugLogging = true;
this._heartbeat.configure(configData);
```

### 1.x Sample Player
...

### Delegates

**1.x:**
- VideoPlayerPluginDelegate()
- VideoPlayerPluginDelegate().getVideoInfo
- VideoPlayerPluginDelegate().getAdBreakInfo
- VideoPlayerPluginDelegate().getAdInfo
- VideoPlayerPluginDelegate().getChapterInfo
- VideoPlayerPluginDelegate().getQoSInfo
- VideoPlayerPluginDelegate().get.onFailure
- AdobeAnalyticsPluginDelegate()
- AdobeHeartbeatPluginDelegate()

**VideoPlayerPluginDelegate:**

```javascript
$.extend(SampleVideoPlayerPluginDelegate.prototype, VideoPlayerPluginDelegate.prototype);
function SampleVideoPlayerPluginDelegate(player) {
    this._player = player;
}
SampleVideoPlayerPluginDelegate.prototype.getVideoInfo =
    function() {
        return this._player.getVideoInfo();
    };
SampleVideoPlayerPluginDelegate.prototype.getAdBreakInfo =
    function() {
        return this._player.getAdBreakInfo();
    };
```

### 2.x Sample Player
...

### 2.x:
- MediaHeartbeatDelegate()
- MediaHeartbeatDelegate().getCurrentPlaybackTime
- MediaHeartbeatDelegate().getQoSObject

**MediaHeartbeatDelegate:**

```javascript
ADB.core.extend(SampleMediaHeartbeatDelegate.prototype, MediaHeartbeatDelegate.prototype);
function SampleMediaHeartbeatDelegate(player) {
    this._player = player;
}
SampleMediaHeartbeatDelegate.prototype.getCurrentPlaybackTime =
    function() {
        return this._player.getCurrentPlaybackTime();
    };
SampleMediaHeartbeatDelegate.prototype.getQoSObject =
    function() {
        return this._player.getQoSObject();
    };
```
### 1.x API

```javascript
SampleVideoPlayerPluginDelegate.prototype.getAdInfo =
    function() {
        return this._player.getAdInfo();
    };

SampleVideoPlayerPluginDelegate.prototype.getChapterInfo =
    function() {
        return this._player.getChapterInfo();
    };

SampleVideoPlayerPluginDelegate.prototype.getQoSInfo =
    function() {
        return this._player.getQoSInfo();
    };
```

### 2.x API

```javascript
this._mediaHeartbeat =
    new MediaHeartbeat(new
        SampleMediaHeartbeatDelegate(this._player),
        mediaConfig,
        appMeasurement);
```

### Sample 2.x Player

... 

### Sample 1.x Player

AdobeAnalyticsPluginDelegate:

```javascript
$.extend(SampleAdobeAnalyticsPluginDelegate.prototype,
    AdobeAnalyticsPluginDelegate.prototype);

function SampleAdobeAnalyticsPluginDelegate() {}{}

SampleAdobeAnalyticsPluginDelegate.prototype.onError =
    function(errorInfo) {
        console.log("AdobeAnalyticsPlugin error: " +
            errorInfo.getMessage() +
            " | " +
            errorInfo.getDetails());
    };
```

### HeartbeatDelegate:

```javascript
$.extend(SampleHeartbeatDelegate.prototype,
    HeartbeatDelegate.prototype);

function SampleHeartbeatDelegate() {}{}

SampleHeartbeatDelegate.prototype.onError =
    function(errorInfo) {
        console.log("Heartbeat error: " +
            errorInfo.getMessage() +
            " | " +
            errorInfo.getDetails());
    };
```

### Sample 1.x Player

Table 50: VHL Code Comparison: CORE PLAYBACK

<table>
<thead>
<tr>
<th></th>
<th>VHL 1.x</th>
<th>VHL 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Session Start</strong></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>VHL 1.x</td>
<td>VHL 2.x</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td><strong>1.x:</strong></td>
<td><strong>2.x:</strong></td>
<td></td>
</tr>
<tr>
<td><em>VideoPlayerPluginDelegate.trackVideoLoad()</em></td>
<td><em>MediaHeartbeat.createMediaObject()</em></td>
<td></td>
</tr>
<tr>
<td><em>VideoPlayerPluginDelegate.getVideoInfo()</em></td>
<td><em>MediaHeartbeat.trackSessionStart()</em></td>
<td></td>
</tr>
</tbody>
</table>

VideoAnalyticsProvider.prototype._onLoad = function() {
  this._playerPlugin.trackVideoLoad();
};
SampleVideoPlayerPluginDelegate.prototype.getVideoInfo = function() {
  return this._player.getVideoInfo();
};

VideoPlayer.prototype.getVideoInfo = function() {
  return this._videoInfo;
};

---

**1.x Sample Player - trackVideoLoad()**

**1.x Sample Player - getVideoInfo()**

---

### Standard Video Metadata

**1.x:**

*VideoMetadataKeys()*

*AdobeAnalyticsPlugin.setVideoMetadata90*

---

**2.x:**

*MediaHeartbeat.createMediaObject()*

---

VideoAnalyticsProvider.prototype._onLoad = function() {
  var contextData = {};
  var videoInfo = this._player.getVideoInfo();
  var mediaInfo = MediaHeartbeat.createMediaObject(
    videoInfo.name,
    videoInfo.id,
    videoInfo.length,
    videoInfo.streamType);
  this._mediaHeartbeat.trackSessionStart(mediaInfo, contextData);
};

---

---

**Sample 1.x Player**

---
### Note: Insetad of setting the Standard Video Metadata through the AdobeAnalyticsPlugin.setVideoMetadata() API, in VHL 2.0, the Standard Video Metadata is set through the MediaObject key MediaObject.MediaObjectKey.StandardVideoMetadata().

#### Custom Video Metadata

**1.x:**
- **VideoMetadataKeys()**
- **AdobeAnalyticsPlugin.setVideoMetadata()**

```javascript
VideoAnalyticsProvider.prototype._onLoad = function() {
    var contextData = {
        isUserLoggedIn: "false",
        tvStation: "Sample TV station",
        programmer: "Sample programmer"
    };
    this._aaPlugin.setVideoMetadata(contextData);
    var videoInfo = this._player.getVideoInfo();
    this._playerPlugin.trackVideoLoad();
    MediaHeartbeat.createMediaObject(videoInfo.name,
```

**2.x:**
- **MediaHeartbeat.createMediaObject()**
- **MediaHeartbeat.trackSessionStart()**

```javascript
VideoAnalyticsProvider.prototype._onLoad = function() {
    var contextData = {
        isUserLoggedIn: "false",
        tvStation: "Sample TV station",
        programmer: "Sample programmer"
    };
    var videoInfo = this._player.getVideoInfo();
    var mediaInfo = MediaHeartbeat.createMediaObject(videoInfo.name,
```
### VHL 1.x

...  

```javascript
...  

```javascript
VideoAnalyticsProvider.prototype._onSeekStart = function() {
    console.log('Player event: SEEK START');
    this._playerPlugin.trackSeekStart();
};

**Sample 1.x Player**  
...

### VHL 2.x

...  

```javascript
...  

```javascript
VideoAnalyticsProvider.prototype._onSeekStart = function() {
    console.log('Player event: SEEK_START');
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekStart);
};

**Sample 2.x Player**  
...

---

**Note:** Instead of setting the Custom Video Metadata through the `AdobeAnalyticsPlugin.setVideoMetadata()` API, in VHL 2.0, the Standard Video Metadata is set through the `MediaHeartbeat.trackSessionStart()` API.

### Playback

**1.x:**  
- `VideoPlayerPlugin.trackPlay()`

**2.x:**  
- `MediaHeartbeat.trackPlay()`

### Pause

**1.x:**  
- `VideoPlayerPlugin.trackPause()`

**2.x:**  
- `MediaHeartbeat.trackPause()`
### VHL 1.x

#### Seek Complete

1.x:

- `VideoPlayerPlugin.trackSeekComplete()`

```javascript
VideoAnalyticsProvider.prototype._onSeekComplete = function() {
  console.log('Player event: SEEK_COMPLETE');
  this._playerPlugin.trackSeekComplete();
};
```

#### Buffer Start

1.x:

- `VideoPlayerPlugin.trackBufferStart()`

```javascript
VideoAnalyticsProvider.prototype._onBufferStart = function() {
  console.log('Player event: BUFFER_START');
  this._playerPlugin.trackBufferStart();
};
```

#### Buffer Complete

1.x:

- `VideoPlayerPlugin.trackBufferComplete()`

```javascript
VideoAnalyticsProvider.prototype._onBufferComplete = function() {
  console.log('Player event: BUFFER_COMPLETE');
  this._playerPlugin.trackBufferComplete();
};
```

### VHL 2.x

#### Sample Player 2.x

#### Seek Complete

2.x:

- `MediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete)`

```javascript
VideoAnalyticsProvider.prototype._onSeekComplete = function() {
  console.log('Player event: SEEK_COMPLETE');
  this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.SeekComplete);
};
```

#### Buffer Start

2.x:

- `MediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferStart)`

```javascript
VideoAnalyticsProvider.prototype._onBufferStart = function() {
  console.log('Player event: BUFFER_START');
  this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferStart);
};
```

#### Buffer Complete

2.x:

- `MediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferComplete)`

```javascript
VideoAnalyticsProvider.prototype._onBufferComplete = function() {
  console.log('Player event: BUFFER_COMPLETE');
  this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BufferComplete);
};
```
### Playback Complete

**1.x:**
- `VideoPlayerPlugin.trackComplete()`

**2.x:**
- `MediaHeartbeat.trackComplete()`

```javascript
VideoAnalyticsProvider.prototype._onComplete = function() {
    console.log('Player event: COMPLETE');
    this._playerPlugin.trackComplete(function() {
        console.log("The completion of the content has been tracked.");
    });
};
```

#### Sample 1.x Player

### Table 51: VHL Code Comparison: AD PLAYBACK

<table>
<thead>
<tr>
<th>VHL 1.x</th>
<th>VHL 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ad Start</strong></td>
<td><strong>Ad Start</strong></td>
</tr>
<tr>
<td><strong>1.x:</strong></td>
<td><strong>2.x:</strong></td>
</tr>
<tr>
<td>• <code>VideoPlayerPlugin.trackAdStart()</code></td>
<td>• <code>MediaHeartbeat.createAdBreakObject()</code></td>
</tr>
<tr>
<td>• <code>VideoPlayerPluginDelegate.getAdBreakInfo()</code></td>
<td>• <code>MediaHeartbeat.createAdObject()</code></td>
</tr>
<tr>
<td>• <code>VideoPlayerPluginDelegate.getAdInfo()</code></td>
<td>• <code>MediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart)</code></td>
</tr>
<tr>
<td>• <code>VideoAnalyticsProvider.prototype._onAdStart = function() {</code></td>
<td>• <code>VideoAnalyticsProvider.prototype._onAdStart = function() {</code></td>
</tr>
</tbody>
</table>
|   console.log('Player event: AD_START');
|   this._playerPlugin.trackAdStart();
|  );
| Sample 1.x Player                   | Sample 1.x Player                   |
|                                       |                                       |
|                                       |                                       |
|                                       |                                       |
|                                       |                                       |
|                                       |                                       |
### VHL 1.x

```
...  
```

### VHL 2.x

```
_@BreakInfo.startTime);

  // Ad Info - getting the adInfo from player
  // and creating
  var _@Info = this._player.getAdInfo();
  var adInfo =
      MediaHeartbeat.createAdObject(_@Info.name,
      _@Info.id,
      _@Info.position,
      _@Info.length);

  this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart,
      adBreakInfo);
  this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart,
      adInfo,
      adContextData);
};
```

## Sample 2.x Player

```
...  
```

### Standard Ad Metadata

#### 1.x:

- `AdMetadataKeys()`
- `AdobeAnalyticsPlugin.setAdMetadata()`

#### 2.x:

- `MediaHeartbeat.createAdObject()`
- `MediaHeartbeat.trackAdStart()`

### VideoAnalyticsProvider.prototype._onAdStart =

```
function() {
    console.log('Player event: AD_START');

    var contextData = {};
    // setting Standard Ad Metadata
    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";
```

### VideoAnalyticsProvider.prototype._onAdStart =

```
function() {
    console.log('Player event: AD_START');

    var adContextData = {};
    // AdBreak Info - getting the adBreakInfo
    // from player and creating
    // AdBreakInfo Object from MediaHeartbeat
    var _adBreakInfo =
        this._player.getAdBreakInfo();
    var adBreakInfo =
        MediaHeartbeat.createAdBreakObject(_adBreakInfo.name,
        _adBreakInfo.position,
        _adBreakInfo.startTime);

    // Ad Info - getting the adInfo from player
    // and creating
    // AdInfo Object from MediaHeartbeat
    var _@Info = this._player.getAdInfo();
```
### VHL 1.x

```javascript
this._aaPlugin.setAdMetadata(contextData);
this._playerPlugin.trackAdStart();
```

### Sample 1.x Player

...

### VHL 2.x

```javascript
var adInfo = MediaHeartbeat.createAdObject(_adInfo.name, _adInfo.id, _adInfo.position, _adInfo.length);
// Set standard Ad Metadata
var standardAdMetadata = {
  [MediaHeartbeat.AdMetadataKeys.ADVERTISER]: "Sample Advertiser",
  [MediaHeartbeat.AdMetadataKeys.CAMPAIGN_ID]: "Sample Campaign",
};
adInfo.setValue(MediaHeartbeat.MediaObjectKey.StandardAdMetadata, standardAdMetadata);
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart, adBreakInfo);
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart, adInfo, adContextData);
```

### Sample 2.x Player

...

---

**Note:** Instead of setting the Standard Ad Metadata through the `AdobeAnalyticsPlugin.setVideoMetadata()` API, in VHL 2.0, the Standard Ad Metadata is set through the `AdMetadata` key `

---

### Custom Ad Metadata

1.x:

- `AdobeAnalyticsPlugin.setAdMetadata()`

2.x:

- `MediaHeartbeat.createAdObject()`
- `MediaHeartbeat.trackAdStart()`

---

```
VideoAnalyticsProvider.prototype._onAdStart = function() {
  console.log('Player event: AD_START');
};
```

```
VideoAnalyticsProvider.prototype._onAdStart = function() {
  console.log('Player event: AD_START');
  var adContextData = {
...```
### VHL 1.x

```javascript
var contextData = {};
// setting Standard Ad Metadata
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);
this._playerPlugin.trackAdStart();
```

**Sample 1.x Player**

...  

### VHL 2.x

```javascript
affiliate: "Sample affiliate",
campaign: "Sample ad campaign";

// AdBreak Info - getting the adBreakInfo
// from player and creating AdBreakInfo Object from MediaHeartbeat
var _adBreakInfo = this._player.getAdBreakInfo();
var adBreakInfo = MediaHeartbeat.createAdBreakObject(_adBreakInfo.name,
  _adBreakInfo.position,
  _adBreakInfo.startTime);

// Ad Info - getting the adInfo from player
// and creating AdInfo Object from MediaHeartbeat
var _adInfo = this._player.getAdInfo();
var adInfo = MediaHeartbeat.createAdObject(_adInfo.name,
  _adInfo.id,
  _adInfo.position,
  _adInfo.length);

this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakStart,
  adBreakInfo);
this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdStart,
  adInfo,
  adContextData);
```

**Sample 2.x Player**

...  

---

**Note:** Instead of setting the Custom Ad Metadata through the `AdobeAnalyticsPlugin.setVideoMetadata` API, in VHL 2.0, the Standard Ad Metadata is set through the `MediaHeartbeat.trackAdStart()` API.

### Ad Skip

<table>
<thead>
<tr>
<th>1.x:</th>
<th>2.x:</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>AdobeAnalyticsPlugin.setAdMetadata()</em></td>
<td><em>MediaHeartbeat.createAdObject()</em></td>
</tr>
</tbody>
</table>
VHL 1.x | VHL 2.x
---|---

SampleVideoPlayerPluginDelegate.prototype.getAdInfo = function() {
    return this._player.getAdInfo();
};

Sample 1.x Player

... 

VideoAnalyticsProvider.prototype._onAdSkip = function() {
    console.log('Player event: AD_SKIP');
    this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdSkip);
};

Sample 1.x Player

... 

Note: In VHL 1.5.X APIs; `getAdInfo()` and `getAdBreakInfo()` must return null if the player is outside the Ad break boundaries.

Ad Complete

1.x:

* VideoPlayerPlugin.trackAdComplete()

Sample 1.x Player

...

2.x:

* MediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdComplete)
* MediaHeartbeat.trackEvent(MediaHeartbeat.Event.AdBreakComplete)

Sample 2.x Player

...

Table 52: VHL Code Comparison: CHAPTER PLAYBACK

<table>
<thead>
<tr>
<th>VHL 1.x</th>
<th>VHL 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chapter Start</strong></td>
<td><strong>Chapter Start</strong></td>
</tr>
<tr>
<td>1.x:</td>
<td>2.x:</td>
</tr>
<tr>
<td>* VideoPlayerPluginDelegate.getChapterInfo()</td>
<td>* MediaHeartbeat.createChapterObject</td>
</tr>
<tr>
<td>* VideoPlayerPlugin.trackChapterStart()</td>
<td>* MediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart)</td>
</tr>
</tbody>
</table>

VideoAnalyticsProvider.prototype._onChapterStart = function() {
    console.log('Player event: CHAPTER_START');
};

VideoAnalyticsProvider.prototype._onChapterStart = function() {
    console.log('Player event: CHAPTER_START');
};
### Chapter Skip

#### 1.x:
- VideoPlayerPluginDelegate.getChapterInfo()

#### 2.x:
- MediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterSkip)

```javascript
SampleVideoPlayerPluginDelegate.prototype.getChapterInfo = function() {
    return this._player.getChapterInfo();
};
```

### Chapter Custom Metadata

#### 1.x:
- VideoPlayerPlugin.trackChapterStart()
- AdobeAnalyticsPlugin.setChapterMetadata()
- VideoAnalyticsProvider.prototype._onChapterStart

#### 2.x:
- MediaHeartbeat.createChapterObject()
- MediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterStart)
- VideoAnalyticsProvider.prototype._onChapterStart

```javascript
VideoAnalyticsProvider.prototype._onChapterStart = function() {
    MediaHeartbeat.trackEvent(MediaHeartbeat.Event.ChapterSkip);
};
```

### Note:
In VHL 1.5.X APIs, `getChapterInfo()` must return null if the player is outside the Chapter boundaries.
function() {
  console.log('Player event: CHAPTER_START');
  this._aaPlugin.setChapterMetadata({
    segmentType: "Sample segment type"
  });
  this._playerPlugin.trackChapterStart();
};

Sample 1.x Player

...
### Table 53: VHL Code Comparison: OTHER EVENTS

<table>
<thead>
<tr>
<th>VHL 1.x</th>
<th>VHL 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bitrate Change</strong></td>
<td><strong>Bitrate Change</strong></td>
</tr>
<tr>
<td><strong>1.x:</strong></td>
<td><strong>2.x:</strong></td>
</tr>
<tr>
<td>• VideoPlayerPlugin.trackBitrateChange()</td>
<td>• MediaHeartbeat.trackEvent(MediaHeartbeat.Event.BitrateChange)</td>
</tr>
<tr>
<td>VideoAnalyticsProvider.prototype._onBitrateChange = function() { console.log('Player event: BITRATE_CHANGE'); // Update getQosInfo to return the updated bitrate this._playerPlugin.trackBitrateChange(); }</td>
<td>VideoAnalyticsProvider.prototype._onBitrateChange = function() { console.log('Player event: BITRATE_CHANGE'); // Update getQosObject to return the updated bitrate this._mediaHeartbeat.trackEvent(MediaHeartbeat.Event.BitrateChange); }</td>
</tr>
</tbody>
</table>

**Sample 2.x Player**

... |

<table>
<thead>
<tr>
<th>VHL 1.x</th>
<th>VHL 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Video Resume</strong></td>
<td><strong>Video Resume</strong></td>
</tr>
<tr>
<td><strong>1.x:</strong></td>
<td><strong>2.x:</strong></td>
</tr>
<tr>
<td>• VideoInfo.resumed()</td>
<td>• MediaObject()</td>
</tr>
<tr>
<td>• VideoPlayerPluginDelegate.getVideoInfo()</td>
<td>• MediaHeartbeat.trackSessionStart()</td>
</tr>
<tr>
<td>• VideoPlayerPlugin.trackVideoLoad()</td>
<td></td>
</tr>
</tbody>
</table>

Sample 1.x Player

... |

<table>
<thead>
<tr>
<th>VHL 1.x</th>
<th>VHL 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>VideoPlayer.prototype.getVideoInfo = function() { this._videoInfo.playhead = vTime; return this._videoInfo; }</td>
<td>VideoAnalyticsProvider.prototype._onLoad = function() { console.log('Player event: MEDIA_LOAD'); var contextData = {}; var videoInfo = this._player.getVideoInfo(); var mediaInfo = MediaHeartbeat.createMediaObject(videoInfo.playerName, videoInfo.id,</td>
</tr>
</tbody>
</table>

**Sample 1.x Player**

... |
### VHL 2.x API References:

- Android API Reference
- iOS API Reference
- JS API Reference

<table>
<thead>
<tr>
<th>Table 54: Required Track APIs:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VHL 1.x</strong></td>
</tr>
<tr>
<td>videoPlayerPlugin.trackVideoLoad()</td>
</tr>
<tr>
<td>videoPlayerPlugin.trackSessionStart()</td>
</tr>
<tr>
<td>videoPlayerPlugin.trackPlay()</td>
</tr>
<tr>
<td>videoPlayerPlugin.trackPause()</td>
</tr>
<tr>
<td>videoPlayerPlugin.trackComplete()</td>
</tr>
<tr>
<td>videoPlayerPlugin.trackVideoUnload()</td>
</tr>
<tr>
<td>videoPlayerPlugin.trackApplicationError()</td>
</tr>
<tr>
<td>videoPlayerPlugin.trackVideoPlayerError()</td>
</tr>
</tbody>
</table>

All of the optional tracking APIs such as (Ads, Chapters, Bitrate change, Seeking, and Buffering) are now part of a single `trackEvent` API. The `trackEvent` API receives a constant parameter that represents the type of event that it is intended to track:

---

For more information on tracking video with 2.x, see *Track Core Video Playback* in *Measuring Video in Adobe Analytics*.
### Table 55: Optional trackEvent APIs:

<table>
<thead>
<tr>
<th>VHL 1.x</th>
<th>VHL 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return a valid AdBreakInfo in</strong> VideoPlayerPlugin.getAdBreakInfo()</td>
<td><code>trackEvent(Event.AdBreakStart)</code></td>
</tr>
<tr>
<td><strong>Return null in</strong> VideoPlayerPlugin.getAdBreakInfo()</td>
<td><code>trackEvent(Event.AdBreakComplete)</code></td>
</tr>
<tr>
<td>playerPlugin.trackAdStart()</td>
<td><code>trackEvent(Event.AdStart, adObject, adCustomMetadata)</code></td>
</tr>
<tr>
<td>playerPlugin.trackAdComplete()</td>
<td><code>trackEvent(Event.AdComplete)</code></td>
</tr>
<tr>
<td><strong>Return null in</strong> VideoPlayerPlugin.getAdInfo()</td>
<td><code>trackEvent(Event.AdSkip)</code></td>
</tr>
<tr>
<td>playerPlugin.trackChapterStart()</td>
<td><code>trackEvent(Event.ChapterStart, chapterObject, chapterCustomMetadata)</code></td>
</tr>
<tr>
<td>playerPlugin.trackChapterComplete()</td>
<td><code>trackEvent(Event.ChapterComplete)</code></td>
</tr>
<tr>
<td><strong>Return null in</strong> VideoPlayerPlugin.getChapterInfo()</td>
<td><code>trackEvent(Event.ChapterSkip)</code></td>
</tr>
<tr>
<td>playerPlugin.trackSeekStart()</td>
<td><code>trackEvent(Event.SeekStart)</code></td>
</tr>
<tr>
<td>playerPlugin.trackSeekComplete()</td>
<td><code>trackEvent(Event.SeekComplete)</code></td>
</tr>
<tr>
<td>playerPlugin.trackBufferStart()</td>
<td><code>trackEvent(Event.BufferStart)</code></td>
</tr>
<tr>
<td>playerPlugin.trackBufferComplete()</td>
<td><code>trackEvent(Event.BufferComplete)</code></td>
</tr>
<tr>
<td>playerPlugin.trackBitrateChange()</td>
<td><code>trackEvent(Event.BitrateChange)</code></td>
</tr>
<tr>
<td>playerPlugin.trackTimedMetadata()</td>
<td><code>trackEvent(Event.TimedMetadataUpdate)</code></td>
</tr>
</tbody>
</table>
Media Collection API

Overview

Introduction
The Media Collection API is Adobe's RESTful alternative to the client-side Media SDK. With the Media Collection API your player can track audio and video events using RESTful HTTP calls. The Media Collection API offers the same realtime tracking of the Media SDK, and in addition provides two additional features:

• **Audio tracking** - Some new audio-specific tracking capabilities are available through the Media Collection API. (This capability will be included in the Media SDK soon.) Audio-specific tracking is implemented through the use of new and updated metadata. In particular, there is a new parameter `streamType`, that identifies whether the stream being tracked is audio or video. Several metrics that were formerly video-oriented are now inclusive of both audio and video. (See [Metrics and metadata](#) for details.) Reporting in Analytics supports audio tracking through reports and an audio consumption template in Workspace (See [Media reports enablement](#) and [Media workspace templates](#) for reporting and workspace updates).

• **Downloaded Content Tracking** - This provides you with the ability to track media while a user is offline, through local storing of event data until the user's device returns online. (See [Track downloaded content](#) for details.)

The Media Collection API is essentially an adapter, acting as a server-side version of the Media SDK. This means that some aspects of the Media SDK documentation (e.g., [Media SDK 2.x](#), [Metrics and Metadata](#)) are also relevant to the Media Collection API.

Media Tracking Data Flows

• A media player implementing the Media Collection API makes RESTful API tracking calls directly to the media tracking back end server, whereas a player implementing the Media SDK makes tracking calls to the SDK APIs inside the player app. One effect of making calls over the web is that the player implementing the Media Collection API needs to handle some of the processing that the Media SDK handles automatically. (Details in [Implementing the API](#).)

• The tracking data captured with the Media Collection API is sent and initially processed differently than the tracking data captured in a Media SDK player, but the same MA processing engine on the backend is used for both solutions.
Media Collection API Overview

**URI:** Obtain this from your Adobe representative.

**HTTP Method:** POST, with JSON request body.

**API Calls:**

- **sessions** - Establishes a session with the server, and returns a Session ID used in subsequent **events** calls. Your app calls this once at the start of a tracking session.
  
  `{uri}/api/v1/sessions`

- **events** - Sends media tracking data.
  
  `{uri}/api/v1/sessions/{session-id}/events`

---

Figure 1: Media Collection API Data Flow
Request Body:

```json
{
    "playerTime": {
        "playhead": {playhead position in seconds},
        "ts": {timestamp in milliseconds}
    },
    "eventType": {event-type},
    "params": {
        {parameter-name}: {parameter-value},
        ...
    },
    "qoeData": {
        {parameter-name}: {parameter-value},
        ...
    },
    "customMetadata": {
        {parameter-name}: {parameter-value},
        ...
    }
}
```

- **playerTime** - Mandatory for all requests.
- **eventType** - Mandatory for all requests.
- **params** - Mandatory for certain eventTypes; check the [JSON validation schema](#) to determine which eventTypes are mandatory, and which are optional.
- **qoeData** - Optional for all requests.
- **customMetadata** - Optional for all requests, but only sent with sessionStart, adStart, and chapterStart event types.

For each eventType, there is a publicly available [JSON validation schema](#) that you should use to verify parameter types and whether a parameter is optional or required for a particular event.

**Event Types:**

- sessionStart
- play
- ping
- pauseStart
- bufferStart
- adStart
- adComplete
- adSkip
- adBreakStart
- adBreakComplete
- chapterStart
- chapterSkip
- chapterComplete
- sessionEnd
- sessionComplete
API reference

Sessions request

POST
http://{uri}/api/v1/sessions

URI Parameters - None

Request Body - Must be JSON, must have the same structure as this sample request body:

```json
{
  "playerTime": {
    "playhead": 0,
    "ts": 1509045324153
  },
  "eventType": "sessionStart",
  "params": {
    "media.playerName": "sample-html5-api-player",
    "analytics.trackingServer": "<your-aa-tracking-server>",
    "analytics.reportSuite": "<your-aa-rsid>",
    "analytics.visitorId": "<your-userId>",
    "media.contentType": "VOD",
    "media.length": 60.39333333333333,
    "media.id": "VA API Sample Player",
    "visitor.marketingCloudOrgId": "<your-org-id>",
    "media.name": "ClickMe",
    "media.channel": "sample-channel",
    "media.sdkVersion": "va-api-0.0.0",
    "analytics.enableSSL": false
  },
  "customMetadata": {
    "myCustomData": "<your metadata>",
    "myCustomData2": "<your metadata>",
    ...
  },
  "qoeData": {
    "param1": "<your param-value>",
    "param2": "<your param-value>",
    ...
  }
}
```

• playerTime (Mandatory)
  • playhead - Must be in seconds, but it can be a float.
  • ts - Timestamp; must be in milliseconds.

• eventType (Mandatory)

• params (Mandatory)

• customMetadata (Optional)

• qoeData (Optional)

Valid eventType value:

• sessionStart

Response -

HTTP/1.1 201 Created
Server: nginx/1.13.5
Date: Wed, 06 Dec 2017 19:14:51 GMT
Table 56: Response Codes

<table>
<thead>
<tr>
<th>HTTP Response Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>Session created</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
</tr>
<tr>
<td>500</td>
<td>Server error</td>
</tr>
</tbody>
</table>

Events request

**POST**

http://(uri)/api/v1/sessions/{sid}/events

**URI Parameter** - sid: Session ID returned from a Sessions request.

**Request Body** - Must be JSON, must have the same structure as this sample request body:

```json
{
    "playerTime": {
        "playhead": 0,
        "ts": 1509045324153
    },
    "eventType": "{event-type}",
    "params": {},
    "qoeData": {},
    "customMetadata": {}
}
```

- **playerTime** (Mandatory)
- **playhead** - Must be in seconds, but it can be a float.
- **ts** - Timestamp; must be in milliseconds.

- **eventType** (Mandatory)
- **params** (Optional)
- **customMetadata** (Optional; send with adStart event type only)
- **qoeData** (Optional)

For a list of valid event types for this release, see Event types and descriptions.
**Important: Ad Tracking** - You can only track ads inside an adBreak.

In the absence of the `adBreakStart` and `adBreakComplete" bookends" around ads, `adStart` and `adComplete` events will simply be ignored, and the corresponding ad duration will be tracked as main content duration. This could have a significant impact on the aggregated data which will be available in Adobe Analytics.

**Response**

HTTP/1.1 204 No Content
Server nginx/1.13.5
Date Thu, 26 Oct 2017 19:15:24 GMT
Connection keep-alive
Access-Control-Allow-Origin *
Access-Control-Allow-Methods OPTIONS,POST,PUT
Access-Control-Allow-Headers Content-Type
Access-Control-Expose-Headers Location

Table 57: HTTP Response Codes

<table>
<thead>
<tr>
<th>HTTP Response Code</th>
<th>Description</th>
<th>Client Action Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>No Content. Heartbeat call was successful.</td>
<td>N/A</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request. Request had improper format.</td>
<td>Check the JSON validation schemas for the request type.</td>
</tr>
<tr>
<td>404</td>
<td>Not Found. The session ID for the media session was not found in the backend service.</td>
<td>The client application should use the Sessions request API to create another media session and report tracking on it.</td>
</tr>
<tr>
<td>410</td>
<td>Gone. The media session was found in the backend service but the client can no longer report activity on it.</td>
<td>The client application should use the Sessions request API to create another media session and report tracking on it.</td>
</tr>
<tr>
<td>500</td>
<td>Server error</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Request parameters**

- page 250
- Additional Details

Table 58: Request Parameter Table

<table>
<thead>
<tr>
<th>Request Key</th>
<th>Required</th>
<th>Set On...</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytics Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>analytics.trackingServer</code></td>
<td>Y</td>
<td><code>sessionStart</code></td>
<td>The URL of your Adobe Analytics server</td>
</tr>
<tr>
<td><code>analytics.reportSuite</code></td>
<td>Y</td>
<td><code>sessionStart</code></td>
<td>The ID that identifies your Analytics reporting data</td>
</tr>
<tr>
<td><code>analytics.enableSSL</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>True or false for enabling SSL</td>
</tr>
<tr>
<td><code>analytics.visitorId</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The Adobe Visitor ID is a custom ID you can use across multiple Adobe</td>
</tr>
<tr>
<td>Request Key</td>
<td>Required</td>
<td>Set On...</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Visitor Data</strong></td>
<td></td>
<td></td>
<td>applications. The Heartbeat <code>visitorId</code> equals the Analytics VID.</td>
</tr>
<tr>
<td><code>visitor.marketingCloudOrgId</code></td>
<td><code>N</code></td>
<td><code>sessionStart</code></td>
<td>The Experience Cloud Organization ID; identifies your organization within the Adobe Experience Cloud eco system.</td>
</tr>
<tr>
<td><code>visitor.marketingCloudUserId</code></td>
<td><code>N</code></td>
<td><code>sessionStart</code></td>
<td>This is the Experience Cloud User ID (ECID). In most scenarios this is the ID you should use to identify a user. The Heartbeat <code>marketingCloudUserId</code> equals the MID in Adobe Analytics. While not technically required, this parameter is necessary for accessing the Experience Cloud family of apps.</td>
</tr>
<tr>
<td><code>visitor.aamLocationHint</code></td>
<td><code>N</code></td>
<td><code>sessionStart</code></td>
<td>Provides Adobe Audience Manager Edge data</td>
</tr>
<tr>
<td><code>appInstallationId</code></td>
<td><code>N</code></td>
<td><code>sessionStart</code></td>
<td>The <code>appInstallationId</code> uniquely identifies the app and the device</td>
</tr>
<tr>
<td><strong>Content Data</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>media.id</code></td>
<td><code>Y</code></td>
<td><code>sessionStart</code></td>
<td>Unique identifier for the content</td>
</tr>
<tr>
<td><code>media.name</code></td>
<td><code>N</code></td>
<td><code>sessionStart</code></td>
<td>Human readable name for the content</td>
</tr>
<tr>
<td><code>media.length</code></td>
<td><code>Y</code></td>
<td><code>sessionStart</code></td>
<td>Content length (seconds)</td>
</tr>
<tr>
<td><code>media.contentType</code></td>
<td><code>Y</code></td>
<td><code>sessionStart</code></td>
<td>Format of the stream (can be any string, a few recommended values are &quot;Live&quot;, &quot;VOD&quot;, or &quot;Linear&quot;)</td>
</tr>
<tr>
<td><code>media.playerName</code></td>
<td><code>Y</code></td>
<td><code>sessionStart</code></td>
<td>The name of the player responsible for rendering the content</td>
</tr>
<tr>
<td><code>media.channel</code></td>
<td><code>Y</code></td>
<td><code>sessionStart</code></td>
<td>The channel of distribution of the content. This could be an mobile application name or a web site name, property name</td>
</tr>
<tr>
<td><code>media.resume</code></td>
<td><code>N</code></td>
<td><code>sessionStart</code></td>
<td>Indicates whether or not to resume a closed session</td>
</tr>
<tr>
<td><code>media.sdkVersion</code></td>
<td><code>N</code></td>
<td><code>sessionStart</code></td>
<td>The SDK version used by the player</td>
</tr>
<tr>
<td><code>media.uniqueTimePlayed</code></td>
<td><code>N</code></td>
<td><code>Close</code></td>
<td>The value in seconds of the unique segments of content played during a session. Excludes time played on seek back scenarios in which a viewer is watching the same segment of the content multiple times.</td>
</tr>
<tr>
<td>Request Key</td>
<td>Required</td>
<td>Set On...</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>media.show</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The program or series name</td>
</tr>
<tr>
<td><code>media.season</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The season number the show or series belongs to</td>
</tr>
<tr>
<td><code>media.episode</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The number of the episode</td>
</tr>
<tr>
<td><code>media.assetId</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The unique identifier for the content of the video asset, such as the TV series episode identifier, movie asset identifier, or live event identifier. Typically these IDs are derived from metadata authorities such as EIDR, TMS/Gracenote, or Rovi. These identifiers can also be from other proprietary or in-house systems.</td>
</tr>
<tr>
<td><code>media.genre</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The type of content as defined by the content producer</td>
</tr>
<tr>
<td><code>media.firstAirDate</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The date when the content first aired on television</td>
</tr>
<tr>
<td><code>media.firstDigitalDate</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The date when the content first aired on any digital platform</td>
</tr>
<tr>
<td><code>media.rating</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The rating as defined by TV Parental Guidelines</td>
</tr>
<tr>
<td><code>media.originator</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The creator of the content</td>
</tr>
<tr>
<td><code>media.network</code></td>
<td>N</td>
<td><code>sessionStart</code></td>
<td>The network / channel name</td>
</tr>
</tbody>
</table>
| `media.showType`  | N        | `sessionStart` | The type of content, expressed as an integer between 0 and 3:  
  • 0 - Full episode  
  • 1 - Preview  
  • 2 - Clip  
  • 3 - Other                                                                                                                                                                                                                                                                                                                                 |
<p>| <code>media.adLoad</code>    | N        | <code>sessionStart</code> | The type of ad loaded                                                                                                                                                                                                                                                                                                                                 |
| <code>media.pass.mvpd</code>  | N        | <code>sessionStart</code> | The MVPD provided by Adobe authentication                                                                                                                                                                                                                                                                                                                                 |
| <code>media.pass.auth</code> | N        | <code>sessionStart</code> | Indicates the user has been authorized by Adobe authentication (can only be true if set)                                                                                                                                                                                                                                                                                                                                 |
| <code>media.dayPart</code>   | N        | <code>sessionStart</code> | The time of day when the content was broadcast                                                                                                                                                                                                                                                                                                                                 |
| <code>media.feed</code>      | N        | <code>sessionStart</code> | The type of feed, e.g., &quot;West-HD&quot;                                                                                                                                                                                                                                                                                                                                 |</p>
<table>
<thead>
<tr>
<th>Request Key</th>
<th>Required</th>
<th>Set On...</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ad Data</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>media.ad.podFriendlyName</td>
<td>N</td>
<td>adBreakStart</td>
<td>Friendly name of the ad break</td>
</tr>
<tr>
<td>media.ad.podIndex</td>
<td>Y</td>
<td>adBreakStart</td>
<td>The index of the ad pod in the video</td>
</tr>
<tr>
<td>media.ad.podSecond</td>
<td>Y</td>
<td>adBreakStart</td>
<td>The second at which the pod started</td>
</tr>
<tr>
<td>media.ad.podPosition</td>
<td>Y</td>
<td>adStart</td>
<td>The index of the ad inside the ad break starting at 1</td>
</tr>
<tr>
<td>media.ad.name</td>
<td>N</td>
<td>adStart</td>
<td>Friendly name of the ad</td>
</tr>
<tr>
<td>media.ad.id</td>
<td>Y</td>
<td>adStart</td>
<td>Name of the ad</td>
</tr>
<tr>
<td>media.ad.length</td>
<td>Y</td>
<td>adStart</td>
<td>Length of the video ad in seconds</td>
</tr>
<tr>
<td>media.ad.playerName</td>
<td>Y</td>
<td>adStart</td>
<td>The name of the player responsible for rendering the ad</td>
</tr>
<tr>
<td><strong>Ad Standard Metadata</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>media.ad.advertiser</td>
<td>N</td>
<td>adStart</td>
<td>The company or brand whose product is featured in the ad</td>
</tr>
<tr>
<td>media.ad.campaignId</td>
<td>N</td>
<td>adStart</td>
<td>The ID of the ad campaign</td>
</tr>
<tr>
<td>media.ad.creativeId</td>
<td>N</td>
<td>adStart</td>
<td>The ID of the ad creative</td>
</tr>
<tr>
<td>media.ad.siteId</td>
<td>N</td>
<td>adStart</td>
<td>The ID of the ad site</td>
</tr>
<tr>
<td>media.ad.creativeURL</td>
<td>N</td>
<td>adStart</td>
<td>The URL of the ad creative</td>
</tr>
<tr>
<td>media.ad.placementId</td>
<td>N</td>
<td>adStart</td>
<td>The placement ID of the ad</td>
</tr>
<tr>
<td><strong>Chapter Data</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>media.chapter.index</td>
<td>Y</td>
<td>chapterStart</td>
<td>Identifies the chapter's position in the content</td>
</tr>
<tr>
<td>media.chapter.offset</td>
<td>Y</td>
<td>chapterStart</td>
<td>The second in the playback where the chapter starts</td>
</tr>
<tr>
<td>media.chapter.length</td>
<td>Y</td>
<td>chapterStart</td>
<td>The length of the chapter in seconds</td>
</tr>
<tr>
<td>media.chapter.friendlyName</td>
<td>N</td>
<td>chapterStart</td>
<td>The human-friendly name of the chapter</td>
</tr>
<tr>
<td><strong>Quality Data</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>media.qoe.bitrate</td>
<td>N</td>
<td>Any</td>
<td>The bitrate of the stream</td>
</tr>
<tr>
<td>media.qoe.bitrateChange</td>
<td>N</td>
<td>Any</td>
<td>The change of the stream bitrate</td>
</tr>
<tr>
<td>media.qoe.droppedFrames</td>
<td>N</td>
<td>Any</td>
<td>The number of dropped frames in the stream</td>
</tr>
<tr>
<td>media.qoe.framesPerSecond</td>
<td>N</td>
<td>Any</td>
<td>The number of frames per second</td>
</tr>
<tr>
<td>media.qoe.timeToStart</td>
<td>N</td>
<td>Any</td>
<td>The amount of time (in milliseconds) passed between when the user hits play and the content loads and starts playing</td>
</tr>
<tr>
<td>media.qoe.errorID</td>
<td>Y</td>
<td>Error</td>
<td>Supports the error event; signals that an error occurred during the session</td>
</tr>
</tbody>
</table>
Additional Details

- **visitor.marketingCloudUserId**

  Pass the Experience Cloud User ID (also known as MID or MCID) on the sessionStart call by including it inside the params map using the following key: `visitor.marketingCloudUserId`. This is a useful feature if you already integrate with other Experience Cloud products and have already obtained the MCID.

  **Note:** Media Analytics is integrated with the Experience Cloud family of apps (Adobe Analytics, Audience Manager, Target, and so on). You need an Experience Cloud ID to access these apps. This is what you should use to identify users in most scenarios.

- **appInstallationId**

  - **If you do not pass an appInstallationId value** - The VA backend will no longer generate a MCID, but instead will rely on Adobe Analytics to do this. Adobe's recommendation is to either send a MCID if available, or an appInstallationId (along with the still mandatory marketingCloudOrgId) so that the Media Collection API generates the MCID and sends it on all calls.

  - **If you do pass appInstallationId value** - The MCID can be generated by the VA back end, if you pass values for appInstallationId and the (required) marketingCloudOrgId parameters. If you do pass appInstallationId yourself, you must persist its value on the client side. It must be unique to the app on a device, and must be persistent for as long as the app is not re-installed.

    **Note:** The appInstallationId uniquely identifies the app and the device. It needs to be unique for each app on each device, i.e., two users using the same version of the same app on different devices must each send a different (unique) appInstallationId.

- **visitor.marketingCloudOrgId**

  In addition to being necessary for MCID generation when that is not provided, this parameter is also used as the value for the publisher ID (based on which Media Analytics performs federation rule matching).

- **Analytics Legacy User ID (aid) and Declared User IDs (customerIDs):**
  
  - **analytics.aid**: The value of this key must be a string that represents the Analytics Legacy User ID
  
  - **visitor.customerIDs**: The value of this key must be an object of the following format:

    ```json
    "<<insert your ID name here>>": { 
    "id": "<<insert your id here>>", 
    "authState": <<insert one of 0, 1, 2>>
    }
    ```

    Note that the **visitor.customerIDs** value can have any number of objects in the presented format.

- **visitor.aamLocationHint**

  AAM Location Hint: This parameter indicates which Adobe Audience Manager (AAM) Edge would be hit when Adobe Analytics sends the customer data to Audience Manager. If you don’t pass this parameter, Adobe hardcodes
it to 1. This is particularly important when end users tend to use their devices in geographically distant locations (e.g., US-East, US-West, Europe, Asia). Otherwise, user data will be spread across multiple AAM Edges.

*media.resume*

If the app determines that a session was closed and then resumed at a later time, e.g., the user left the video but eventually came back, and the player resumed the video from the playhead where it was stopped, you can send an optional boolean `media.resume` parameter inside the params bucket of the `sessionStart` call.

---

**Event types and descriptions**

- **sessionStart** - Sent with the `sessions` call. When the response returns, you extract the Session ID from the Location header and use it for subsequent event calls to the Collection server.
- **play** - Sent when the player changes state to "playing" from another state (i.e., the `on('Playing')` callback is triggered by the player). Other states from which the player moves to "playing" include "buffering", the user resumed from "paused", the player recovered from an error, autoplay, and so on.
- **ping**:
  - **Main Content** - Must be sent every 10 seconds during main content playback, regardless of other API events that have been sent. The first ping event should fire 10 seconds after main content playback has begun.
  - **Ad Content** - Must be sent every 1 second during ad tracking.

Ping events should not include the `params` map in the request body.

- **bufferStart** - Sent when buffering starts. There is no `bufferResume` event type. A `bufferResume` is inferred when you send a `play` event after `bufferStart`.
- **pauseStart** - Sent when the user presses Pause. There is no `resume` event type. A `resume` is inferred when you send a `play` event after `pauseStart`.
- **adBreakStart** - Signals the start of an ad break
- **adStart** - Signals the start of an ad
- **adComplete** - Signals the completion of an ad break
- **adSkip** - Signals an ad skip
- **adBreakComplete** - Signals the completion of an ad break
- **chapterStart** - Signals the start of a chapter segment
- **chapterStart** - Signals a chapter skip
- **chapterComplete** - Signals the completion of a chapter
- **sessionEnd** - This is used to notify the VA backend to immediately close the session when the user has abandoned viewing the content and is unlikely to return

If you don't send a `sessionEnd`, an abandoned session will time-out normally (after no events are received for 10 minutes, or when no playhead movement occurs for 30 minutes), and the session is deleted by the backend.

- **sessionComplete** - Sent when the end of the main content is reached

---

**Important:** You should refer to the JSON validation schemas for each event type, to verify correct event parameter types and requirements.
JSON validation schemas

The VA backend validates the request parameters for each event type using JSON validation schemas. These schemas are available to you, and serve as the current authority on parameter types used in the VA API.

POST
http://{uri}/api/v1/schemas/{event-type}

For more information about using the JSON validation schemas, see Validating event requests.

Implementing the API

Quick start

Tip: Gather the request data necessary for completing a successful Sessions request to the Media Collection server. You can quickly verify your request data by sending requests manually (with curl, or Postman, etc.). This will give you immediate feedback on whether you have any issues with incorrect data types or incorrect information in your request. Use the JSON validation schemas to verify that you are supplying proper request data.

1. Gather the standard, required Adobe Analytics and Visitor data that you must supply to run any of the Experience Cloud applications:
   • Visitor Experience Cloud Org ID
   • Visitor Experience Cloud User ID (ECID)
   • Analytics Report Suite ID
   • Analytics Tracking Server URL

2. Create a JSON object for your sessions request body, containing the minimum data required for a successful call. For example:

   ```json
   {
      "playerTime": {
        "playhead": 0,
        "ts": 1234560890123
      },
      "eventType": "sessionStart",
      "params": {
        "media.playerName": "sample-html5-api-player",
        "analytics.trackingServer": "[YOUR_TS]",
        "analytics.reportSuite": "[YOUR_RSID]",
        "media.contentType": "VOD",
        "media.length": 60.39333333333333,
        "media.id": "VA API Sample Player",
        "visitor.marketingCloudOrgId": "[YOUR_ORG_ID]",
        "visitor.marketingCloudUserId": "[YOUR_ECID]",
        "media.name": "ClickMe",
        "media.channel": "sample-channel",
        "media.sdkVersion": "va-api-0.0.0",
        "analytics.enableSSL": false
      }
   }
   ```
3. Send sessions requests to the VA API endpoint. If your request payload is invalid, identify the problem and retry until you get a 201 Created response. In this curl example, the JSON request body is in a file named sample_data_session:

```bash
$ curl -i -d @sample_data_session http://{uri}/api/v1/sessions 
> curl.sessions.out

$ cat curl.sessions.out
HTTP/1.1 201 Created
Server: nginx/1.13.5
Date: Mon, 18 Dec 2017 22:34:12 GMT
Content-Type: application/octet-stream
Content-Length: 0
Connection: keep-alive
Location: /api/v1/sessions/\[\[\[<== Session ID ]
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: OPTIONS,POST,PUT
Access-Control-Allow-Headers: Content-Type
Access-Control-Expose-Headers: Location
```

If the Sessions request succeeds, you receive a 201 Created response similar to the one above. The response includes a Session ID in the Location header. The Session ID is the crucial piece of information in the response, as it is required for all subsequent tracking calls. After a successful return of a Sessions request, you can confidently proceed with implementing video tracking using the VA API in your video player.

### Setting the HTTP request type in your player

The request body for all Media Collection API requests must be in JSON format, so you should set the content request type in your player. For example, in JavaScript you would set the Content-Type request header as follows:

```javascript
httpRequest.setRequestHeader('Content-Type', 'application/json');
```

### Obtaining a session ID

This code snippet from the Reference Player shows one way of coding a Sessions request, along with extracting the Session ID (and the Collection API version) from the Location header in the response:

```javascript
var sessionData = {
    ...
    "media.contentType": "VOD",
    "media.channel": "sample-channel",
    ...
};

const SESSION_ID_EXTRACTOR = /^/api/(.*)/sessions/(.*)/;

apiClient.request({
    "baseUrl": config.apiBaseUrl, // The endpoint
    "path": config.apiSessionsPath, // api/v1/sessions/
    "method": "POST", // (Always POST)
    "data": sessionData // Mandatory params
}).then((response) => {
    // Extract Session ID (and API version)
```
### Implementing an events request

**{uri}/api/v1/sessions/{sid}/events** - Use the *Events request* for all subsequent tracking calls after you obtain a Session ID using the *Sessions request*. Specify the playhead location and timestamp, the event type, along with any optional parameters you want to include, in the JSON body of the request.

The JSON request body for the *Events request* has the same structure as that of the *sessions request*, however check the *JSON validation schemas* for parameter requirements and types.

### Validating event requests

The JSON request body for each event type is validated on the backend with JSON schemas. The HTTP response body is populated with an error message when validation fails for an API call.

JSON validation schemas for each event type are publicly accessible here: `{uri}/api/v1/schemas/{eventType}`, e.g., `{uri}/api/v1/schemas/sessionEnd`. These JSON validation schemas are the absolute authority for determining the current and correct request body parameters and their data types for each type of event.

For example, the response to a request for the *sessionStart* validation schema appears something like this sample (slightly formatted for readability here):

```
HTTP/1.1 200 OK
Server: nginx/1.13.5
Date: Thu, 18 Jan 2018 15:44:50 GMT
Content-Type: application/json
Content-Length: 2716
Connection: keep-alive

{"$schema":"http://json-schema.org/draft-04/schema#","id":"http://alpha.hb-api.omtrdc.net/api/v1/schemas/sessionStart","definitions":{"playerTime":{"type":"object","properties":{"playhead":{"type":"number"},"ts":{"type":"integer"}},"required":["playhead","ts"],"additionalProperties":false},"eventType":{"type":"string","enum":["sessionStart","play","ping","bufferStart","pauseStart","sessionComplete","bufferEnd","sessionStart","sessionComplete","bitrateChange","error","adBreakStart","adBreakComplete","adStart","adComplete","adSkip","sessionEnd"],"qoeData":{"type":"object","properties":{"media.qoe.bitrate":{"type":"integer"},"media.qoe.droppedFrames":{"type":"integer"},"media.qoe.framesPerSecond":{"type":"integer"},"media.qoe.timeToStart":{"type":"integer"},"required":[],"additionalProperties":false},"customMetadata":{"type":"object","patternProperties":{"^[a-zA-Z0-9_.]+$":{"type":"string"},"additionalProperties":false},"sessionStart":{"properties":{"eventType":{"type":"string","enum":["sessionStart"]}}},
```

```
"playerTime":{"$ref":"#/definitions/playerTime"},
"params":{
  "type": "object",
  "properties": {
    "appInstallationId": {
      "type": "string"
    },
    "analytics.trackingServer": {
      "type": "string"
    },
    "analytics.reportSuite": {
      "type": "string"
    },
    "visitor.marketingCloudOrgId"
  },
  "additionalProperties": false
},
"customMetadata":{
  "$ref": "/#definitions/customMetadata"
},
"qoeData":{
  "$ref": "/#definitions/qoeData"
}

**Note:** Session level validation is not possible, as the session context is not available in the collection layer.

### Sending ping events

*For main content, you must fire ping events every 10 seconds*, beginning after 10 seconds of playback, regardless of other API events that you have sent. For Ad tracking, you must fire ping events every 1 second.

- This is literally the "heartbeat" of Media Analytics. The only required parameters for a ping call are `eventType: ping` along with the `playerTime` object (playhead position and timestamp). The following code snippet shows one way to implement a timed pinging mechanism for main content (10 second interval):

```javascript
...  Pinger.init(10000);
...
  Pinger.kill();

var Pinger = {
  init: function(interval) {
    this._timer = window.setInterval(function() {
      $.event.trigger({
        type: "onPing",
        _data: ""
      });
    }, interval);
  },
  kill: function() {
    window.clearInterval(this._timer);
  }
};
```

### Sending QoE data

Every event can be decorated with an extra JSON key called `qoeData`, which is positioned alongside the `params` key in the JSON request body.

**Note:** You should check the **JSON validation schemas** to verify parameter types and whether they are mandatory or optional.

### Custom metadata support

You can provide custom key/value pairs on the `sessionStart` and `adStart` events. This information must be provided in the JSON key, `customMetadata`, positioned alongside the `params` key.
The customMetadata JSON key should contain an object of key/value pairs. The key should contain only alphanumerical characters, underline, and dot/period.

**Timeout conditions**

**Media Collection API Timeout Conditions:** The Media Collection API, being stateless, does not have the same mechanism as the Media SDK for issuing a new Session ID when timeout conditions occur. When a timeout condition occurs, the back end will close the session, and all subsequent calls made with that Session ID will be dropped. The logic that handles a Session Timeout must be handled in the client. That is, the player will have to monitor the timeout conditions, and obtain a new Session ID if a timeout occurs.

- **10 Minutes: No API Events** - If the back end does not receive any API events it will close the session.
- **30 Minutes: No Playhead Change** - If the playhead does not move for 30 minutes (e.g., the user hits Pause and walks away), the back end will close the session.

💡 **Note:** You can also force a session end by sending an events request with the sessionEnd event type.

**Controlling the order of events**

Since the VA API is RESTful, and video tracking is a highly time-dependent operation, an implementor could be concerned about VA API tracking calls arriving at the back end out of order. The back end does attempt to queue up and reorder events based on the provided timestamp in the playerTime object. However, there is a limit to this capability. Currently, the reorder may fail if the delays between out of order calls are more than one second. This "acceptable delay time" may be optimized and / or be configurable in future updates.

**Queueing events when sessions response is slow**

The Media Collection API is RESTful: i.e., you make an HTTP request and wait for the response. This is only important when you make a Sessions request to obtain a Session ID at the beginning of video playback. Your player may fire events before the sessions response returns (with the Session ID parameter) from the backend. In this case your app must queue any tracking events that arrive between the Sessions request and its response. When the sessions response arrives, you should first process any queued events, then you can start processing live events with the events API. The Events request does not return data back to the client beyond an HTTP response code. Check the Reference Player in your distribution for one way to process events prior to receiving a Session ID. For example:

```javascript
var eventData = {};  // JSON payload
eventData.playerTime = getPlayerTime();  // Required
eventData.eventType = "play";  // Required
eventData.params = {};  // Optional for events

VideoPlayer.prototype._collectEvent = function(eventData) {
  // If we don't have a Session ID yet, queue the event and return...
  if (!sessionStarted) {
    console.log("[Player] Queueing event ");
    _pendingEvents.push(eventData);
    return;
  }
  // If we DO have a Session ID, process the tracking event...
  apiClient.request({
```

---
VideoPlayer.prototype.collectEvent =
function (eventType, eventParams) {
    if (typeof eventParams === 'undefined') {
        eventParams = {};
    }
    this._collectEvent(
        eventType: eventType,            // Required
        playerTime: getPlayerTime(),     // Required
        params: eventParams              // Optional
    );
};

VideoPlayer.prototype.getPlayerTime = function() {
    return {
        playhead: this.getPlayhead(),    // playhead value in seconds
        ts: this.getCurrentTimestamp()    // timestamp value in milliseconds
    };
};

Process any queued events - The reference player processes queued events as follows:

continue to process tracking events as they occur.
Media tracking timelines

Timeline 1 - View to end of content

Playhead Timeline

![Playhead Timeline Diagram]

Actions Timeline

![Actions Timeline Diagram]

Table 59: VOD, pre-roll ads, pausing, buffering, viewing content to the end

<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Action Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
</table>
| 1        | Auto-play or Play button pressed, video | 0                          | 0                          | /api/v1/sessions {
|          |        |                           |                            | playerTime: {
|          |        |                           |                            | player: 0,
|          |        |                           |                            | ts: <timestamp>
|          |        |                           |                            | },
|          |        |                           |                            | eventType: 'sessionStart',
|          |        |                           |                            | params: {
|          |        |                           |                            | "media.playerName": "sample-html5-api-player",
<p>|          |        |                           |                            | } | This call signals the user's intention to play a video. It returns a Session ID ((sid)) to the client |</p>
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action Description</th>
<th>Action Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>starts loading.</td>
<td></td>
<td></td>
<td></td>
<td>that is used to identify all subsequent tracking calls within the session. The player state is not yet &quot;playing&quot;, but is instead &quot;starting&quot;. Mandatory session parameters must be included in the params map in the request body. On the backend, this call generates an Adobe Analytics initiate call.</td>
</tr>
<tr>
<td>2</td>
<td>App starts ping event timer</td>
<td>0</td>
<td>0</td>
<td></td>
<td>Start your app's 10-second ping timer. First ping event should then fire 10 seconds into the session.</td>
</tr>
<tr>
<td>3</td>
<td>Track pre-roll ad break start</td>
<td>0</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Ads can only be tracked within an ad break.</td>
</tr>
<tr>
<td>4</td>
<td>Track pre-roll Ad #1 start</td>
<td>0</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Start tracking the first pre-roll ad, which is 15 seconds long. Including custom metadata with this adStart.</td>
</tr>
<tr>
<td>Action #</td>
<td>Action</td>
<td>Action Timeline (Seconds)</td>
<td>Playhead position (Seconds)</td>
<td>Client Request</td>
<td>Implementation Details</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------</td>
<td>---------------------------</td>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.creativeURL&quot;: &quot;http://xyz_creative.com&quot;, &quot;media.ad.placementId&quot;: &quot;sample_placement&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>customMetadata: { &quot;myCustomData1&quot;: &quot;CustomData1&quot;, &quot;myCustomData2&quot;: &quot;CustomData2&quot; }</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Ad pings</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>App sends ping event</td>
<td>1</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: ping }</td>
<td>Ping the backend every 1 second.</td>
</tr>
<tr>
<td>5.2</td>
<td>App sends ping event</td>
<td>2</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: ping }</td>
<td>Ping the backend every 1 second.</td>
</tr>
<tr>
<td>5.3</td>
<td>App sends ping event</td>
<td>3</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: ping }</td>
<td>Ping the backend every 1 second.</td>
</tr>
<tr>
<td>6</td>
<td>Track pre-roll Ad #1 complete</td>
<td>15</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: adComplete }</td>
<td>Track the end of the first pre-roll ad.</td>
</tr>
<tr>
<td>7</td>
<td>Track pre-roll Ad #2 start</td>
<td>15</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: adStart, }</td>
<td>Track the start of the second pre-roll ad, which is 7 seconds long.</td>
</tr>
</tbody>
</table>

💡 Note: Subsequent ads in the timeline will skip showing the series of one-second pings in the interest of brevity...
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Action Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
</table>
| 8       | App sends ping event | 20 | 0 | /api/v1/sessions/{sid}/events { 
  playerTime: 
  player: 0,
  ts: <timestamp>
}, 
  eventType:ping | Ping the backend every 10 seconds. |
| 9       | Track pre-roll Ad #2 complete | 22 | 0 | /api/v1/sessions/{sid}/events { 
  playerTime: 
  player: 0,
  ts: <timestamp>
}, 
  eventType:adComplete | Track the end of the second pre-roll ad. |
| 10      | Track pre-roll ad break complete | 22 | 0 | /api/v1/sessions/{sid}/events { 
  playerTime: 
  player: 0,
  ts: <timestamp>
}, 
  eventType:adBreakComplete | The ad break is over. Throughout the ad break, the play state has remained "playing". |
| 11      | Track play event | 22 | 0 | /api/v1/sessions/{sid}/events { 
  playerTime: 
  player: 0,
  ts: <timestamp>
}, 
  eventType:play | After the adBreakComplete event, put the player is in the "playing" state. |
| 12      | App sends ping event | 30 | 8 | /api/v1/sessions/{sid}/events { 
  playerTime: 
  player: 8,
  ts: <timestamp>
}, 
  eventType:ping | Ping the backend every 10 seconds. |
| 13      | Buffer start event occurred | 33 | 11 | /api/v1/sessions/{sid}/events { 
  playerTime: 
  player: 11,
  ts: <timestamp>
}, | Track the player's move to the "buffering" state. |

```javascript
params: {
  "media.ad.podFriendlyName": "ad_pod1",
  "media.ad.name": "Ad 2",
  "media.ad.id": "002",
  "media.ad.length": 7,
  "media.ad.podPosition": 1,
  "media.ad.playerName": "Sample Player",
  "media.ad.advertiser": "Ad Guys",
  "media.ad.campaignId": "*",
  "media.ad.creativeId": "44",
  "media.ad.siteId": "XYZ",
  "media.ad.creativeURL": "http://xyz_creative.com",
  "media.ad.placementId": "sample_placement2"
},
```

Ping the backend every 10 seconds.
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Action Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Buffering ends after 3 seconds, so put the player back to the &quot;playing&quot; state. You must send another track play event coming out of buffering. <strong>The play call after a bufferStart infers a &quot;bufferEnd&quot; call to the back end</strong>, so there is no need for a bufferEnd event.</td>
</tr>
<tr>
<td>14</td>
<td>Buffering ended, the app tracks resumption of content</td>
<td>36</td>
<td>11</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 11, ts: &lt;timestamp&gt; }, eventType:play }</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>App sends ping event</td>
<td>40</td>
<td>15</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 15, ts: &lt;timestamp&gt; }, eventType:ping }</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td>16</td>
<td>Track mid-roll ad break start</td>
<td>46</td>
<td>21</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 21, ts: &lt;timestamp&gt; }, eventType:adBreakStart, params: { &quot;media.ad.podFriendlyName&quot;: &quot;ad_pod2&quot;, &quot;media.ad.podIndex&quot;: 1, &quot;media.ad.podSecond&quot;: 21 } }</td>
<td>Mid-roll ad of 8 seconds duration: send adBreakStart.</td>
</tr>
<tr>
<td>17</td>
<td>Track mid-roll Ad #3 start</td>
<td>46</td>
<td>21</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 21, ts: &lt;timestamp&gt; }, eventType:adStart, params: { &quot;media.ad.podFriendlyName&quot;: &quot;ad_pod2&quot;, &quot;media.ad.name&quot;: &quot;Ad 3&quot;, &quot;media.ad.id&quot;: &quot;003&quot;, &quot;media.ad.length&quot;: 8, &quot;media.ad.podPosition&quot;: 2, &quot;media.ad.playerName&quot;: &quot;Sample Player&quot;, &quot;media.ad.advertiser&quot;: &quot;Ad Guys&quot;, &quot;media.ad.campaignId&quot;: &quot;7&quot;, &quot;media.ad.creativeId&quot;: &quot;40&quot;, &quot;media.ad.siteId&quot;: &quot;XYZ&quot;, &quot;media.ad.creativeURL&quot;: [266] }</td>
<td>Track the mid-roll ad.</td>
</tr>
<tr>
<td>Action #</td>
<td>Action</td>
<td>Action Timeline (Seconds)</td>
<td>Playhead position (Seconds)</td>
<td>Client Request</td>
<td>Implementation Details</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>---------------------------</td>
<td>-----------------------------</td>
<td>----------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;http://xyz_creative.com&quot;, &quot;media.ad.placementId&quot;: &quot;sample_placement2&quot;</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td>18</td>
<td>App sends ping event</td>
<td>50</td>
<td>21</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 21, ts: &lt;timestamp&gt; }, eventType:ping }</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td>19</td>
<td>Track mid-roll Ad #1 complete</td>
<td>54</td>
<td>21</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 21, ts: &lt;timestamp&gt; }, eventType:adComplete }</td>
<td>The mid-roll ad is complete.</td>
</tr>
<tr>
<td>20</td>
<td>Track mid-roll ad break complete</td>
<td>54</td>
<td>21</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 21, ts: &lt;timestamp&gt; }, eventType:adBreakComplete }</td>
<td>The ad break is complete.</td>
</tr>
<tr>
<td>21</td>
<td>App sends ping event</td>
<td>60</td>
<td>27</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 27, ts: &lt;timestamp&gt; }, eventType:ping }</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td>22</td>
<td>User pressed Pause</td>
<td>64</td>
<td>31</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 31, ts: &lt;timestamp&gt; }, eventType:pauseStart }</td>
<td>The user's action moves the play state to &quot;paused&quot;.</td>
</tr>
<tr>
<td>23</td>
<td>App sends ping event</td>
<td>70</td>
<td>31</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 31, ts: &lt;timestamp&gt; }, eventType:ping }</td>
<td>Ping the backend every 10 seconds. Player is still in the &quot;buffering&quot; state; the user is stuck at 20 seconds of content. Fuming...</td>
</tr>
</tbody>
</table>
### Action Details

<table>
<thead>
<tr>
<th>Action #</th>
<th>Action Description</th>
<th>Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>User pressed Play to resume main content</td>
<td>74</td>
<td>31</td>
<td><code>/api/v1/sessions/{sid}/events</code>&lt;br&gt;<code>&lt;br&gt;  playerTime:&lt;br&gt;      playerhead: 31,&lt;br&gt;      ts: &lt;timestamp&gt;&lt;br&gt;    eventType:play&lt;br&gt;`&lt;br&gt;</code></td>
<td>Move the play state to &quot;playing&quot;. The <em>play</em> call after a <em>pauseStart</em> infers a &quot;resume&quot; call to the back end, so there is no need for a <em>resume</em> event.</td>
</tr>
<tr>
<td>25</td>
<td>App sends ping event</td>
<td>80</td>
<td>37</td>
<td><code>/api/v1/sessions/{sid}/events</code>&lt;br&gt;<code>&lt;br&gt;  playerTime:&lt;br&gt;      playerhead: 37,&lt;br&gt;      ts: &lt;timestamp&gt;&lt;br&gt;    eventType:ping&lt;br&gt;`&lt;br&gt;</code></td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td>26</td>
<td>The user finishes watching the content to the end.</td>
<td>88</td>
<td>45</td>
<td><code>/api/v1/sessions/{sid}/events</code>&lt;br&gt;<code>&lt;br&gt;  playerTime:&lt;br&gt;      playerhead: 45,&lt;br&gt;      ts: &lt;timestamp&gt;&lt;br&gt;    eventType:sessionComplete&lt;br&gt;`&lt;br&gt;</code></td>
<td>Send <code>sessionComplete</code> to the back end to indicate that the user finished watching the entire content.</td>
</tr>
</tbody>
</table>

**Note:** No Seek Events? - There is no explicit support in the Media Collection API for *seekStart* or *seekComplete* events. This is because certain players generate a very large number of such events when the end-user is scrubbing, and several hundred users could easily bottleneck the network bandwidth of a backend service. Adobe works around explicit support for seek events by computing heartbeat duration based on device timestamp, rather than playhead position.
Timeline 2 - User abandons session

Playhead Timeline

Actions Timeline
Table 60: VOD, Pre-roll ad, mid-roll ads, user abandons content early

<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Action Timeline (Seconds)</th>
<th>Playhead Position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
</table>
| 1        | Auto-play or Play button pressed            | 0                         | 0                          | /api/v1/sessions
{
  playerTime:
    playhead: 0,
    ts: <timestamp>
},
  eventType: sessionStart,
  params:
    "media.playerName": "sample-html5-api-player",
    "analytics.trackingServer": "[YOUR_TS]",
    "analytics.reportSuite": "[YOUR_RSID]",
    "analytics.visitorId": "[YOUR_VISITOR_ID]",
    "analytics.enableSSL": false
}                                                                                           | This call signals the user's intention to play a video. It returns a Session ID to the client that is used to identify all subsequent tracking calls within the session. The player state is not yet "playing", but is instead "starting". Mandatory session parameters must be included in the params map in the request body. On the backend, this call generates an Adobe Analytics initiate call. |
| 2        | App starts ping event timer                 | 0                         | 0                          | start your app's 10-second ping timer. First ping event should then fire 10 seconds into the session. |
| 3        | Track pre-roll ad break start               | 0                         | 0                          | /api/v1/sessions/{sid}/events
{
  playerTime:
    playhead: 0,
    ts: <timestamp>,
  eventType: adBreakStart,
  params:
    "media.ad.podFriendlyName": "ad_pod1",
    "media.ad.podIndex": 0,
    "media.ad.podSecond": 0
}                                                                                           | Pre-roll ads must be tracked. Ads can only be tracked within an ad break.                                                                   |
| 4        | Track pre-roll Ad #1 start                  | 0                         | 0                          | /api/v1/sessions/{sid}/events
{
  playerTime:
    playhead: 0,
    ts: <timestamp>
},
  eventType: midRollAdStart
}                                                                                           | A 12 second ad starts.                                                                                                                      |
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action Description</th>
<th>Timeline (Seconds)</th>
<th>Playhead Position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
</table>
| 5       | Ad pings           | 1                  | 0                           | /api/v1/sessions/{sid}/events
```
{
  playerTime:
  {
    playhead: 0,
    ts: <timestamp>
  },
  eventType: ping
}
``` |
| 6       | Track pre-roll Ad #1 complete | 12                | 0                           | /api/v1/sessions/{sid}/events
```
{
  playerTime:
  {
    playhead: 0,
    ts: <timestamp>
  },
  eventType: adComplete
}
``` |
| 7       | Track pre-roll ad break complete | 12                | 0                           | /api/v1/sessions/{sid}/events
```
{
  playerTime:
  {
    playhead: 0,
    ts: <timestamp>
  },
  eventType: adBreakComplete
}
``` |
| 8       | Track play event   | 12                | 0                           | /api/v1/sessions/{sid}/events
```
{
  playerTime:
  {
    playhead: 0,
    ts: <timestamp>
  },
  eventType: play,
  qoeData: {
    bitrate: 10000
  }
}
``` |
| 9       | App sends ping event | 20                | 8                           | /api/v1/sessions/{sid}/events
```
{
  playerTime:
  {
    playhead: 88,
    ts: <timestamp>
  },
``` |

- eventTypes: `adStart`,
- params: 
  - `media.ad.podFriendlyName`: "ad_pod1",
  - `media.ad.name": "Ad 1",
  - `media.ad.id": "002",
  - `media.ad.length": 7,
  - `media.ad.podPosition": 1,
  - `media.ad.playerName": "Sample Player",
  - `media.ad.advertiser": "Ad Guys",
  - `media.ad.campaignId": "1",
  - `media.ad.creativeId": "42",
  - `media.ad.siteId": "XYZ",
  - `media.ad.creativeURL": "http://xyz.creative.com",
  - `media.ad.placementId": "sample_placement2"

- Ping the backend every 1 second.

Note: (Subsequent ad pings not shown in the interest of brevity.)
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Action Timeline (Seconds)</th>
<th>Playhead Position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>App sends ping event</td>
<td>30</td>
<td>18</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{ playerTime: { playhead: 18, ts: &lt;timestamp&gt; }, eventType: ping }</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Error occurs, app sends error information.</td>
<td>32</td>
<td>20</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>For error events, qoeData parameters errorID and errorSource are required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{ playerTime: { playhead: 20, ts: &lt;timestamp&gt; }, eventType: error, qoeData: { &quot;media.qoe.errorID&quot;: &quot;&lt;errorID&gt;&quot;, &quot;media.qoe.media.errorSource&quot;: &quot;player&quot; } }</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>App recovers from error, user presses Play</td>
<td>37</td>
<td>20</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{ playerTime: { playhead: 18, ts: &lt;timestamp&gt; }, eventType: play, qoeData: { bitrate: 10000 } }</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>App sends ping event</td>
<td>40</td>
<td>28</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{ playerTime: { playhead: 28, ts: &lt;timestamp&gt; }, eventType: ping }</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Track mid-roll ad break start</td>
<td>45</td>
<td>33</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Mid-roll ad of 8 seconds duration: send adBreakStart.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{ playerTime: { playhead: 33, ts: &lt;timestamp&gt; }, eventType: adBreakStart, params: { &quot;media.ad.podFriendlyName&quot;: &quot;ad_pod2&quot;, &quot;media.ad.podIndex&quot;: 1, &quot;media.ad.podSecond&quot;: 33 } }</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Track mid-roll Ad #1 start</td>
<td>45</td>
<td>33</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Track the mid-roll ad.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{ playerTime: { playhead: 33, ts: &lt;timestamp&gt; }, eventType: adStart, params: { &quot;media.ad.podFriendlyName&quot;: &quot;ad_pod1&quot;, &quot;media.ad.name&quot;: &quot;Ad 1&quot; } }</td>
<td></td>
</tr>
<tr>
<td>Action #</td>
<td>Action Description</td>
<td>Timeline Position (Seconds)</td>
<td>Client Request</td>
<td>Implementation Details</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------------------</td>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------</td>
<td></td>
</tr>
</tbody>
</table>
| 16      | User closes the app. The app determines that the user has abandoned viewing and isn't returning to this session. | 48                         | ```json
"media.ad.id": "002",
"media.ad.length": 8,
"media.ad.podPosition": 1,
"media.ad.playerName": "Sample Player",
"media.ad.advertiser": "Ad Guys",
"media.ad.campaignId": "7",
"media.ad.creativeId": "40",
"media.ad.siteId": "XYZ",
"media.ad.creativeURL": "http://xyz_creative.com",
"media.ad.placementId": "sample_placement2"
} ``` | Send sessionEnd to the VA backend to indicate that the session should be closed immediately, with no further processing. |

---

**Timeline 3 - Chapters**

**Playhead Timeline**

```
Ad Break
Ad 1, Ad 2

0
Δ
Buffer
Chapter 1

Ad Break
Ad 3

11
22
Δ
Pause
Chapter 2

31
45
```
### Actions Timeline

<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Action Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
</tr>
</thead>
</table>
| 1        | Auto-play or Play button pressed, video starts loading. | 0 | 0 | /api/v1/sessions

```json
{
  playerTime: {
    playhead: 0,
    ts: <timestamp>
  },
  eventType: sessionStart,
  params: {
    "media.playerName": "sample-html5-api-player",
    "analytics.trackingServer": "+[YOUR_TS]",
    "analytics.reportSuite": "+[YOUR_RSID]",
    "analytics.visitorId": "+[YOUR_VISITOR_ID]",
    "media.contentType": "VOD",
    "media.length": 60.3333333333333,
    "media.id": "VA API Sample Player",
    "visitor.marketingCloudOrgId": "[YOUR_MCID]*",
    "media.name": "ClickMe",
    "media.channel": "sample-channel",
    "media.sdkVersion": "va-api-0.0.0",
    "analytics.enableSSL": false
  }
}
```

The call signals the user's intention to play a video. It returns a Session ID ([sid]) to the client that is used to identify all subsequent tracking calls within the session. The player state is not yet "playing", but instead "starting". Mandatory session parameters must be included in the params map in the request body.

On the backend, this call generates an Adobe Analytics initiate call.
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>App starts ping event timer</td>
<td>0</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: adBreakStart, params: { &quot;media.ad.podFriendlyName&quot;: &quot;ad_pod1&quot;, &quot;media.ad.podIndex&quot;: 0, &quot;media.ad.podSecond&quot;: 0 } }</td>
<td>Start your app’s 10-second ping timer. First ping event should then fire 10 seconds into the session.</td>
</tr>
<tr>
<td>3</td>
<td>Track pre-roll ad break start</td>
<td>0</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: adBreakStart, params: { &quot;media.ad.podFriendlyName&quot;: &quot;ad_pod1&quot;, &quot;media.ad.podIndex&quot;: 0, &quot;media.ad.podSecond&quot;: 0 } }</td>
<td>Ads can only be tracked within an ad break.</td>
</tr>
<tr>
<td>4</td>
<td>Track pre-roll Ad #1 start</td>
<td>0</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: adStart, params: { &quot;media.ad.podFriendlyName&quot;: &quot;ad_pod1&quot;, &quot;media.ad.name&quot;: &quot;Ad 1&quot;, &quot;media.ad.id&quot;: &quot;001&quot;, &quot;media.ad.length&quot;: 15, &quot;media.ad.podPosition&quot;: 1, &quot;media.ad.playerName&quot;: &quot;Sample Player&quot;, &quot;media.ad.advertiser&quot;: &quot;Ad Guys&quot;, &quot;media.ad.campaignId&quot;: &quot;1&quot;, &quot;media.ad.creativeId&quot;: &quot;42&quot;, &quot;media.ad.siteId&quot;: &quot;XYZ&quot;, &quot;media.ad.creativeURL&quot;: &quot;http://xyz_creative.com&quot;, &quot;media.ad.placementId&quot;: &quot;sample_placement&quot; }, customMetadata: { &quot;myCustomData1&quot;: &quot;CustomData1&quot;, &quot;myCustomData2&quot;: &quot;CustomData2&quot; } }</td>
<td>Start tracking the first pre-roll ad, which is 15 seconds long. Including custom metadata with this adStart.</td>
</tr>
<tr>
<td>5</td>
<td>Ad pings</td>
<td>1</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: ping }</td>
<td>Ping the backend every 1 second.</td>
</tr>
</tbody>
</table>

*Note:* (Subsequent ad pings not shown in the interest of brevity.)
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Track pre-roll Ad #1 complete</td>
<td>15</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events {playerTime: {playhead: 0, ts: &lt;timestamp&gt;}, eventType:adComplete}</td>
<td>Track the end of the first pre-roll ad.</td>
</tr>
<tr>
<td>7</td>
<td>Track pre-roll Ad #2 start</td>
<td>15</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events {playerTime: {playhead: 0, ts: &lt;timestamp&gt;}, eventType:adStart, params: {&quot;media.ad.podFriendlyName&quot;: &quot;ad_pod1&quot;, &quot;media.ad.name&quot;: &quot;Ad 2&quot;, &quot;media.ad.id&quot;: &quot;002&quot;, &quot;media.ad.length&quot;: 7, &quot;media.ad.podPosition&quot;: 1, &quot;media.ad.playerName&quot;: &quot;Sample Player&quot;, &quot;media.ad.advertiser&quot;: &quot;Ad Guys&quot;, &quot;media.ad.campaignId&quot;: &quot;2&quot;, &quot;media.ad.creativeId&quot;: &quot;44&quot;, &quot;media.ad.siteId&quot;: &quot;XYZ&quot;, &quot;media.ad.creativeURL&quot;: &quot;http://xyz_creative.com&quot;, &quot;media.ad.placementId&quot;: &quot;sample_placement2&quot;}},</td>
<td>Track the start of the second pre-roll ad, which is 7 seconds long.</td>
</tr>
<tr>
<td>8</td>
<td>Ad pings</td>
<td>16</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events {playerTime: {playhead: 0, ts: &lt;timestamp&gt;}, eventType:ping}</td>
<td>Ping the backend every 1 second.</td>
</tr>
<tr>
<td>9</td>
<td>Track pre-roll Ad #2 complete</td>
<td>22</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events {playerTime: {playhead: 0, ts: &lt;timestamp&gt;}, eventType:adComplete}</td>
<td>Track the end of the second pre-roll ad.</td>
</tr>
<tr>
<td>10</td>
<td>Track pre-roll ad break complete</td>
<td>22</td>
<td>0</td>
<td>/api/v1/sessions/{sid}/events {playerTime: {playhead: 0, ts: &lt;timestamp&gt;}, eventType:adBreakComplete}</td>
<td>The ad break is over. Throughout the ad break, the play state has remained &quot;playing&quot;.</td>
</tr>
<tr>
<td>Action #</td>
<td>Action</td>
<td>Action Timeline (Seconds)</td>
<td>Playhead position (Seconds)</td>
<td>Client Request</td>
<td>Implementation Details</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>---------------------------</td>
<td>-----------------------------</td>
<td>---------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>11</td>
<td>Track play event</td>
<td>22</td>
<td>0</td>
<td><code>/api/v1/sessions/{sid}/events</code> { <code>playerTime:</code>{ <code>playhead</code>: 0, <code>ts</code>: &lt;timestamp&gt; }, <code>eventType</code>:<code>play</code> }</td>
<td>After the adBreakComplete event, put the player is in the &quot;playing&quot; state using the play event.</td>
</tr>
<tr>
<td>12</td>
<td>Track chapter start event</td>
<td>23</td>
<td>1</td>
<td><code>/api/v1/sessions/{sid}/events</code> { <code>playerTime:</code>{ <code>playhead</code>: 0, <code>ts</code>: &lt;timestamp&gt; }, <code>eventType</code>:<code>chapterStart</code>, <code>params</code>: { <code>&quot;media.chapter.index&quot;:</code> 1, <code>&quot;media.chapter.offset&quot;:</code> 0, <code>&quot;media.chapter.length&quot;:</code> 20, <code>&quot;media.chapter.friendlyName&quot;: &quot;Chapter Uno&quot;</code> }, }</td>
<td>After the play event, track the start of the first chapter.</td>
</tr>
<tr>
<td>13</td>
<td>App sends ping event</td>
<td>30</td>
<td>8</td>
<td><code>/api/v1/sessions/{sid}/events</code> { <code>playerTime:</code>{ <code>playhead</code>: 8, <code>ts</code>: &lt;timestamp&gt; }, <code>eventType</code>:<code>ping</code> }</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td>14</td>
<td>Buffer start event occurred</td>
<td>33</td>
<td>11</td>
<td><code>/api/v1/sessions/{sid}/events</code> { <code>playerTime:</code>{ <code>playhead</code>: 11, <code>ts</code>: &lt;timestamp&gt; }, <code>eventType</code>:<code>bufferStart</code> }</td>
<td>Track the player's move to the &quot;buffering&quot; state.</td>
</tr>
</tbody>
</table>
| 15       | Buffering ended, the app tracks resumption of content | 36 | 11 | `/api/v1/sessions/{sid}/events` { `playerTime:`{ `playhead`: 11, `ts`: <timestamp> }, `eventType`:`play` } | Buffering ends after 3 seconds, so put the player back to the "playing" state. You must send another track play event coming out of buffering.  
*The play call after a bufferStart infers a "bufferEnd" call to the back end, so there is no need for a bufferEnd event.*** |
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action Timeline (Seconds)</th>
<th>Action (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td>40 15</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ playerTime:{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>playhead: 15,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ts: &lt;timestamp&gt; }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>eventType:ping</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>45 20</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>The first chapter ends, right before the second ad break.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ playerTime:{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>playhead: 20,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ts: &lt;timestamp&gt; }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>eventType:chapterEnd</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>46 21</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Mid-roll ad of 8 seconds duration: send adBreakStart.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ playerTime:{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>playhead: 21,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ts: &lt;timestamp&gt; }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>eventType:adBreakStart,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>params: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.podFriendlyName&quot;: &quot;ad_pod2&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.podIndex&quot;: 1,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.podSecond&quot;: 21</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>46 21</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Track the mid-roll ad.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ playerTime:{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>playhead: 21,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ts: &lt;timestamp&gt; }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>eventType:adStart,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>params: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.podFriendlyName&quot;: &quot;ad_pod2&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.name&quot;: &quot;Ad 3&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.id&quot;: &quot;003&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.length&quot;: 8,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.podPosition&quot;: 2,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.playerName&quot;: &quot;Sample Player&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.advertiser&quot;: &quot;Ad Guys&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.campaignId&quot;: &quot;7&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.creativeId&quot;: &quot;40&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.siteId&quot;: &quot;XYZ&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.creativeURL&quot;: &quot;http://xyz_creative.com&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;media.ad.placementId&quot;: &quot;sample_placement2&quot;</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>47 21</td>
<td>/api/v1/sessions/{sid}/events</td>
<td>Ping the backend every 1 second.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ playerTime:{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>playhead: 21,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ts: &lt;timestamp&gt; }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>eventType:ping</td>
<td></td>
</tr>
<tr>
<td>Action #</td>
<td>Action Description</td>
<td>Action Timeline (Seconds)</td>
<td>Playhead Position (Seconds)</td>
<td>Client Request</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------</td>
<td>--------------------------</td>
<td>-----------------------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| 21      | Track mid-roll Ad #1 complete | 54                       | 21                          | /api/v1/sessions/{sid}/events { 
  playerTime: {
    playhead: 21,
    ts: <timestamp>
  },
  eventType: adComplete
} | The mid-roll ad is complete. |
| 22      | Track mid-roll ad break complete | 54                       | 21                          | /api/v1/sessions/{sid}/events { 
  playerTime: {
    playhead: 22,
    ts: <timestamp>
  },
  eventType: adBreakComplete
} | The ad break is complete. |
| 23      | Track the start of Chapter 2 | 55                       | 22                          | /api/v1/sessions/{sid}/events { 
  playerTime: {
    playhead: 22,
    ts: <timestamp>
  },
  eventType: chapterStart,
  params: {
    "media.chapter.index": 2,
    "media.chapter.offset": 22,
    "media.chapter.length": 22,
    "media.chapter.friendlyName": "Chapter Dos"
  }
} | |
| 24      | App sends ping event | 60                       | 27                          | /api/v1/sessions/{sid}/events { 
  playerTime: {
    playhead: 27,
    ts: <timestamp>
  },
  eventType: ping
} | Ping the backend every 10 seconds. |
| 25      | User pressed Pause | 64                       | 31                          | /api/v1/sessions/{sid}/events { 
  playerTime: {
    playhead: 31,
    ts: <timestamp>
  },
  eventType: pauseStart
} | The user's action moves the play state to "paused". |
| 26      | App sends ping event | 70                       | 31                          | /api/v1/sessions/{sid}/events { 
  playerTime: {
    playhead: 31,
    ts: <timestamp>
  },
  eventType: ping
} | Ping the backend every 10 seconds. Player is still in the "buffering" state; the user is stuck at 20 seconds of content. Fuming... |
<table>
<thead>
<tr>
<th>Action #</th>
<th>Action</th>
<th>Timeline (Seconds)</th>
<th>Playhead position (Seconds)</th>
<th>Client Request</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>User pressed Play to resume main content</td>
<td>74</td>
<td>31</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 31, ts: &lt;timestamp&gt; }, eventType: play }</td>
<td>Move the play state to “playing”. The play call after a pauseStart infers a “resume” call to the back end, so there is no need for a resume event.</td>
</tr>
<tr>
<td>28</td>
<td>App sends ping event</td>
<td>80</td>
<td>37</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 37, ts: &lt;timestamp&gt; }, eventType: ping }</td>
<td>Ping the backend every 10 seconds.</td>
</tr>
<tr>
<td>29</td>
<td>Chapter 2 ends</td>
<td>87</td>
<td>44</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 0, ts: &lt;timestamp&gt; }, eventType: chapterEnd }</td>
<td>Track the end of the second and final chapter.</td>
</tr>
<tr>
<td>30</td>
<td>The user finishes watching the content to the end.</td>
<td>88</td>
<td>45</td>
<td>/api/v1/sessions/{sid}/events { playerTime: { playhead: 45, ts: &lt;timestamp&gt; }, eventType: sessionComplete }</td>
<td>Send sessionComplete to the backend to indicate that the user finished watching the entire content.</td>
</tr>
</tbody>
</table>

*Note: No Seek Events?* - There is no explicit support in the Media Collection API for seekStart or seekComplete events. This is because certain players generate a very large number of such events when the end-user is scrubbing, and several hundred users could easily bottleneck the network bandwidth of a backend service. Adobe works around explicit support for seek events by computing heartbeat duration based on device timestamp, rather than playhead position.

### Track downloaded content

#### Overview
The Downloaded Content API provides the ability to track media consumption while a user is offline. For example, a user downloads and installs an app on a mobile device. The user then downloads content using the app into local storage on the device. To track this downloaded data, Adobe has developed the Downloaded Content API. With
this API, when the user plays content from a device's storage, tracking data is stored on the device, regardless of the device's connectivity. When the user finishes the playback session, and the device is online, the stored tracking information is sent to the Media Collection API backend inside a single payload. From there, processing and reporting proceeds as normal in the Media Collection API, on which this API is based.

Contrast the realtime approach of the Media Collection API with the batch processing approach of the Downloaded Content API. With the realtime approach, the media player sends tracking data upon each player event, and it sends network pings every ten seconds (every one second for ads), one by one to the backend. With Downloaded Content, the same session events need to be generated but stored on the device until sent as a single session (see example below). Each approach has its advantages and disadvantages: the Media Collection API tracks in realtime, but this requires a connectivity check before each network call; the Downloaded Content API only needs one network connectivity check, but also requires a larger memory footprint on the device.

Implementation

Event schemas: The Downloaded Content API is based on the Media Collection API, so the event data that your player batches and sends requires that the same events schemas are used as in the Media Collection API. For information on these schemas, see: Overview; and: Validating event requests.

Order of events:

• The first event in the batch payload must be sessionStart.
• You must include media.downloaded: true in the standard metadata parameters (params key) on the sessionStart event. If this parameter is not present or is set to false, the Downloaded Content API will return a 400 response code (Bad Request). This is so the backend can distinguish between downloaded and live content, and process it accordingly. (Note that if media.downloaded: true is set on a live session, this will likewise result in a 400 response from the Media Collection API.)
• It is the responsibility of the implementation to correctly store player events in the order of their appearance.

Response codes:

• 201 - Created: Successful Request; the data is valid and the session was created and will be processed.
• 400 - Bad Request; schema validation has failed, all data is discarded, no sessions data will be processed.

Integration with Adobe Analytics

When computing the Analytics start/close calls for the downloaded content scenario, the backend sets an extra Analytics field called ts. These are timestamps for the first and last events received (initiate and complete). This mechanism allows a completed media session to be placed at the correct point in time (i.e., even if the user doesn't come back online for several days, the media session is reported to have occurred at the time the content was actually viewed). You must enable this mechanism on the Adobe Analytics side by creating a timestamp optional report suite. To enable a timestamp optional report suite, see Timestamps Optional.

Sample session comparison

[url]/v1/sessions

• Media Collection API:

```json
{
  eventType: "sessionStart",
  playerTime: {
    playhead: 0,
    ts: 1529997923478
  },
  params: { /* Standard metadata parameters as documented */ },
}"
• Downloaded Content API:

```json
[
    {
        "eventType": "sessionStart",
        "playerTime": {"playhead": 0, "ts": 1529997923478},
        "params": {
            "media.downloaded": true
        },
        "customMetadata": {},
        "qoeData": {}},
    {
        "eventType": "play",
        "playerTime": {"playhead": 0, "ts": 1529997928174},
        "params": {}
    },
    {
        "eventType": "ping",
        "playerTime": {"playhead": 10, "ts": 1529997937503},
        "params": {}
    },
    {
        "eventType": "ping",
        "playerTime": {"playhead": 20, "ts": 1529997947533},
        "params": {}
    },
    {
        "eventType": "ping",
        "playerTime": {"playhead": 30, "ts": 1529997957545},
        "params": {}
    },
    {
        "eventType": "sessionComplete",
        "playerTime": {"playhead": 35, "ts": 1529997960559}
    }
]
```
Metrics and metadata

Audio and video parameters

⚠️ **Important:** On February 7th, 2019, Adobe Analytics for Video and Audio released a metric name change. Media Initiates will now be called Media Starts. This change was made to reflect industry standards in metrics and reporting, and to make the metric easily identifiable in reporting.

💡 **Note:** On September 13th, 2018 a change was made to the labels for some dimensions, metrics, and reports, to allow for cross-content tracking of video and audio analytics. The labels that have been changed include: video initiates to media initiates, video length to content length, and video name to content name. The video reports in Reports and Analytics have all been updated to use the name "Media" rather than "Video". The label changes did not change data collection or historic data. Please take note of these changes in the case that you're using them within Report Builder or in any other external automated data pulls that might be impacted by this change.

This topic presents a list of audio and video content data, including context data values, that Adobe collects via solution variables.

- Core Audio and Video Data
- Standard Audio and Video Metadata
- Audio and Video Metrics

Table data description:

- **Label:** The name of the parameter.
- **Implementation:** Information on implementation values and requirements
  - **Key:** Variable, set either manually in your app, or automatically by the Adobe Media SDK.
  - **Required:** Indicates whether the parameter is required for basic audio and video tracking.
  - **Type:** Specifies the type of the variable to be set, string or number.
  - **Sent With:** Indicates when the data is sent: *Initiate* is the analytics call sent on media start, *Ad Start* is the analytics call sent on ad start, *Chapter Start* is the analytics call sent on chapter start, and *Close* is the compiled analytics call sent directly from the heartbeat server to the analytics server at the end of the media session, or the end of the ad. The Close calls are not available in network packet calls.
  - **Min. SDK Version:** Indicates which SDK version you would need to access the parameter.
  - **Sample Value:** Provides example of common variable usage.
- **Network Parameters:** Displays the values that are passed to Adobe Analytics or Heartbeat servers. This column shows the names of the parameters that are seen in the network calls generated by Adobe Media SDKs.
  - **Available:** Indicates whether the data is available in reporting by default (Yes), or requires custom set-up (Custom)
  - **Reserved Variable:** Indicates whether the data is captured as an event, eVar, prop, or classification in a reserved variable.
  - **Expiration:** Indicates whether the data expires after each hit or after each visit.
  - **Report Name:** Name of Adobe Analytics report for variable
  - **Context Data:** Name of the Adobe Analytics context data passed to the reporting server and used in processing rules.
Data Feed - Column name for variable found in Clickstream or Live Stream data feeds

Audience Manager - Trait name found in Adobe Audience Manager

**Important:** Do not change the classification names for any variables listed below that are described under Reporting/Reserved Variable as "classification".

The media classifications are defined when a report suite is enabled for media tracking. From time to time, Adobe adds new properties, and, when this occurs, customers must re-enable their report suites to get access to the new media properties. During the update process Adobe determines whether the classifications are enabled by checking the names of the variables. If any of them is missing, Adobe adds the missing ones again.

## Core Audio and Video Data

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream Type</td>
<td>• SDK Key: streamType* &lt;br&gt;• API Key: media.streamType &lt;br&gt;• Required: Yes &lt;br&gt;• Type: string &lt;br&gt;• Sent with: Media Start, Media Close &lt;br&gt;• Min. SDK Version: 1.5 &lt;br&gt;• Sample value: “video”</td>
<td>• Adobe Analytics: a.media.streamType &lt;br&gt;• Heartbeats: s:meta:a.media.streamType</td>
<td>• Available: Yes &lt;br&gt;• Reserved Variable: eVar &lt;br&gt;• Expiration: On VISIT &lt;br&gt;• Report Name: Content &lt;br&gt;• Context Data: a.media.streamType &lt;br&gt;• Data Feed: videostreamtype &lt;br&gt;• Audience Manager: c_contextdata.a.media.streamType</td>
</tr>
<tr>
<td>Content ID</td>
<td>• SDK Key: mediaId* &lt;br&gt;• API Key: media_id &lt;br&gt;• Required: Yes &lt;br&gt;• Type: string &lt;br&gt;• Sent with: Media Start, Media Close</td>
<td>• Adobe Analytics: a.media.name &lt;br&gt;• Heartbeats: s:asset:video_id</td>
<td>• Available: Yes &lt;br&gt;• Reserved Variable: eVar &lt;br&gt;• Expiration: On VISIT &lt;br&gt;• Report Name: Content &lt;br&gt;• Context Data: a.media.name &lt;br&gt;• Data Feed: video</td>
</tr>
</tbody>
</table>

Release Date: 09/13/18

*Note:* Available in Media Collection API or Media SDK 2.2.

Identifies the stream type. Valid values are "audio", "video", and "".

**Segments:**

- StreamType "All" - Segment all media stream data. **Rule:** Content (ID) exists
- StreamType "Audio" - Segment all audio stream data. **Rule:** Content (ID) exists AND Stream Type = audio
- StreamType "Video" - Segment all video stream data. **Rule:** Content (ID) exists AND Stream Type = video
### Reporting Network Parameters

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• <strong>Min. SDK Version:</strong> Any</td>
<td>• <strong>Audience Manager:</strong> c_contextdata.a.media.name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Sample value:</strong> &quot;4586695ABC&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content Length (variable)</td>
<td>• <strong>SDK Key:</strong> length*</td>
<td>• <strong>Adobe Analytics:</strong> a.media.length</td>
<td>• <strong>Available:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>• <strong>API Key:</strong> media.length</td>
<td>• <strong>Heartbeats:</strong> l:asset:length</td>
<td>• <strong>Reserved Variable:</strong> eVar</td>
</tr>
<tr>
<td></td>
<td>• <strong>Required:</strong> Yes</td>
<td></td>
<td>• <strong>Expiration:</strong> On HIT</td>
</tr>
<tr>
<td></td>
<td>• <strong>Type:</strong> number</td>
<td></td>
<td>• <strong>Report Name:</strong> Content Length (variable)</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sent with:</strong> Media Start, Media Close</td>
<td></td>
<td>• <strong>Context Data:</strong> a.media.length</td>
</tr>
<tr>
<td></td>
<td>• <strong>Min. SDK Version:</strong> Any</td>
<td></td>
<td>• <strong>Data Feed:</strong> videolength</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sample value:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• VOD: 128</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Live: 86400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Linear: 1800</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Release Date:</strong> 09/13/18</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Clip Length/Runtime - This is the maximum length (or duration) of the content being consumed (in seconds). It equals the last value of l:asset:length from events of type Main. If l:asset:length is not set, then the last value of l:asset:duration is used. In reporting, Video Length is the classification, and Content Length (variable) is the eVAR.

**Important:** This property is used to compute several metrics, such as progress tracking metrics and Average Minute Audience. If this is not set, or not greater than zero, then these metrics are not available.

For Live media with an unknown duration, the value of 86400 is the default.

Pre Version 1.5.1, this was l:asset:duration; after 1.5.1, this is l:asset:length.

### Video Length

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• <strong>SDK Key:</strong> length*</td>
<td>• <strong>Adobe Analytics:</strong> a.media.length</td>
<td>• <strong>Available:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>• <strong>API Key:</strong> media.length</td>
<td>• <strong>Heartbeats:</strong> l:asset:length</td>
<td>• <strong>Reserved Variable:</strong> Classification</td>
</tr>
<tr>
<td></td>
<td>• <strong>Required:</strong> Yes</td>
<td></td>
<td>• <strong>Expiration:</strong> On HIT</td>
</tr>
<tr>
<td></td>
<td>• <strong>Type:</strong> number</td>
<td></td>
<td>• <strong>Report Name:</strong> Video Length</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sent with:</strong> Media Start, Media Close</td>
<td></td>
<td>• <strong>Context Data:</strong> a.media.length</td>
</tr>
<tr>
<td></td>
<td>• <strong>Min. SDK Version:</strong> Any</td>
<td></td>
<td>• <strong>Data Feed:</strong> videoclassificationlength</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sample value:</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


### Content Type

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• VOD: 128</td>
<td></td>
<td>• Audience Manager: c_contextdata.a.media.length</td>
</tr>
<tr>
<td></td>
<td>• Live: 86400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Linear: 1800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Release Date: 09/27/18**

Clip Length/Runtime - This is the maximum length (or duration) of the content being consumed (in seconds). It equals the last value of `l:asset:length` from events of type Main. If `l:asset:length` is not set, then the last value of `l:asset:duration` is used.

In reporting, Video Length is the classification, and Content Length (variable) is the eVAR.

**Important:** This property is used to compute several metrics, such as progress tracking metrics and Average Minute Audience. If this is not set, or not greater than zero, then these metrics are not available.

For Live media with an unknown duration, the value of 86400 is the default.

Pre Version 1.5.1, this was `l:asset:duration`; after 1.5.1, this is `l:asset:length`.

### Available values per Stream Type:

- **Audio:** "song", "podcast", "audiobook", "radio"
- **Video:** "VoD", "Live", "Linear", "UGC", "DVoD"

Customers can provide custom values for this parameter.

This equals `s:stream:type`. If that is unset, this equals `missing_content_type`.

### Media session ID

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• SDK Key: Automatically set</td>
<td></td>
<td>• Available: Use processing rule</td>
</tr>
<tr>
<td></td>
<td>• API Key: Obtained from backend</td>
<td></td>
<td>• Reserved Variable: N/A</td>
</tr>
<tr>
<td></td>
<td>• Required: Yes</td>
<td></td>
<td>• Report Name: Custom</td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td></td>
<td>• Context Data: <code>a.media.vsid</code></td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Start, Media Close</td>
<td></td>
<td>• Data Feed: <code>vsid</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Audience Manager: c_contextdata.a.media.vsid</td>
</tr>
<tr>
<td>Label</td>
<td>Implementation</td>
<td>Network Parameters</td>
<td>Reporting</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>--------------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
|               | • Min. SDK Version: 1.5.8  
|               | • Sample value: 1482236761294786918253 | | |
| This identifies an instance of a content stream unique to an individual playback. |

### Content Player Name
- **SDK Key:** playerName+
- **API Key:** media.playerName
- **Required:** Yes
- **Type:** string
- **Sent with:** Media Start, Media Close
- **Min. SDK Version:** Any
- **Sample value:** "Brightcove", "Primetime", etc.
- **Adobe Analytics:**
  - a.media.playerName
  - **Heartbeats:**
  - s:sp:player_name
- **Available:** Yes
- **Reserved Variable:** eVar
- **Expiration:** On HIT
- **Report Name:** Content Player Name
- **Context Data:**
  - a.media.playerName
- **Data Feed:** videoplayername
- **Audience Manager:**
  - c_contextdata.a.media.playerName

Name of the player.

### Content Channel
- **SDK Key:** channel+
- **API Key:** media.channel
- **Required:** Yes
- **Type:** string
- **Sent with:** Media Start, Media Close
- **Min. SDK Version:** Any
- **Sample value:** "Sports"
- **Adobe Analytics:**
  - a.media.channel
  - **Heartbeats:**
  - s:sp:channel
- **Available:** Yes
- **Reserved Variable:** eVar
- **Expiration:** On HIT
- **Report Name:** Content Channel
- **Context Data:**
  - a.media.channel
- **Data Feed:** videochannel
- **Audience Manager:**
  - c_contextdata.a.media.channel

Distribution Station/Channels or where the content is played. Any string value is accepted here.

### Content Segment
- **SDK Key:** Automatically set
- **API Key:** N/A
- **Required:** Yes
- **Type:** string
- **Sent with:** Media Close
- **Min. SDK Version:** Any
- **Sample value:** "[0-10]" (minutes)
- **Adobe Analytics:** N/A
  - **Heartbeats:** N/A
- **Available:** Yes
- **Reserved Variable:** eVar
- **Expiration:** On HIT
- **Report Name:** Content Segment
- **Context Data:**
  - a.media.segment
- **Data Feed:** videsegment
- **Audience Manager:**
  - c_contextdata.a.media.segment
The interval that describes the part of the content that has been viewed (in minutes). The segment is computed as min and max of the playhead values during a playback session.

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content Name (variable)</strong></td>
<td>• SDK Key: name*</td>
<td>• Adobe Analytics: a.media.friendlyName</td>
<td>• Available: Yes</td>
</tr>
</tbody>
</table>
  • API Key: media.name | • Heartbeats: s:asset:name | • Reserved Variable: eVar |  
  • Required: No |  
  • Type: string |  
  • Sent with: Media Start, Media Close | • Expiration: On HIT |  
  • Min. SDK Version: 1.5.1 | • Report Name: Content Name (variable) |  
  • Sample value: “The Big Bang Theory” | • Context Data: a.media.friendlyName |  
  | • Adobe Analytics: a.media.friendlyName | • Data Feed: videoname |  
  | • Heartbeats: s:asset:name | • Audience Manager: c_contextdata.a.media.friendlyName |  
| **Release Date: 09/13/18** |  
  In reporting, Video Name is the classification, and Content Name (variable) is the eVAR. |  
  This is the “friendly” (human-readable) name of the content, equal to the last value of s:asset:name. |  
| **Video Name** | • SDK Key: name* | • Adobe Analytics: a.media.friendlyName | • Available: Yes |  
  • API Key: media.name | • Heartbeats: s:asset:name | • Reserved Variable: Classification |  
  • Required: No |  
  • Type: string |  
  • Sent with: Media Start, Media Close | • Expiration: On HIT |  
  • Min. SDK Version: 1.5.1 | • Report Name: Video Name |  
  • Sample value: “The Big Bang Theory” | • Context Data: a.media.friendlyName |  
  | • Adobe Analytics: a.media.name | • Data Feed: videoclassificationname |  
  | • Heartbeats: s:asset:video_id | • Audience Manager: c_contextdata.a.media.friendlyName |  
| **Release Date: 09/27/18** |  
  In reporting, Video Name is the classification, and Content Name (variable) is the eVAR. |  
  This is the “friendly” (human-readable) name of the content, equal to the last value of s:asset:name. |  
| **Video Path** | • SDK Key: Automatically set | • Adobe Analytics: a.media.name | • Available: Yes |  
  • API Key: N/A | • Heartbeats: s:asset:video_id | • Reserved Variable: prop |  
  • Required: No |  
  • Type: string |  
  • Sent with: Initiate | • Report Name: Video Path |  
<p>| • Adobe Analytics: a.media.name | • Context Data: a.media.name |<br />
| • Heartbeats: s:asset:video_id | • Data Feed: videopath |</p>
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. SDK Version</td>
<td>Any</td>
<td></td>
<td>Audience Manager: c_contextdata.a.media.name</td>
</tr>
<tr>
<td>Sample value</td>
<td>&quot;4586695ABC&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ability to track path of viewer across site and/or App to see path they took to view a particular video. Any integer and/or letter combination.

|                        | API Key: media.sdkVersion                                                     | Heartbeats: s:sp:/sdk                                                             | Reserved Variable: N/A                                                      |
|                        | Required: No                                                                  |                                                                                    | Report Name:                                                               |
|                        | Type: string                                                                  |                                                                                    | Context Data: a.media.sdkVersion`                                           |
|                        | Sent with: Media Close                                                        |                                                                                    | Data Feed: N/A                                                            |
|                        | Min. SDK Version: 1.5.7                                                       |                                                                                    | Audience Manager: c_contextdata.a.media.sdkVersion                         |
|                        | Sample value: "2.62.0_release"                                                 |                                                                                    |                                                                           |

The SDK version used by the player. This could have any custom value that makes sense for your player. Customers will have to create their own processing rules to have the value available for reporting.

|                        | API Key: N/A                                                                  | Heartbeats: s:sp:hb_version                                                       | Reserved Variable: N/A                                                      |
|                        | Required: No                                                                  |                                                                                    | Report Name:                                                               |
|                        | Type: string                                                                  |                                                                                    | Context Data: a.media.vhlVersion`                                           |
|                        | Sent with: Media Close                                                        |                                                                                    | Data Feed: N/A                                                            |
|                        | Min. SDK Version: 1.5.7                                                       |                                                                                    | Audience Manager: c_contextdata.a.media.vhlVersion                         |
|                        | Sample value: "js-2.0.1.88-c8c0b1"                                            |                                                                                    |                                                                           |

The heartbeat SDK version used for the tracking session. Customers will have to create their own processing rules to have the value available for reporting.

```javascript
MediaHeartbeat.version();
```

- `createMediaObject` APIs:
  - Android - `createMediaObject`
  - iOS - `createMediaObjectWithName`
  - JavaScript - `createMediaObject`
  - Chromecast - `createMediaObject`
- `MediaHeartbeatConfig` APIs:
  - Android - `MediaHeartbeatConfig`
**Standard Audio and Video Metadata**

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>SDK Key:</strong> SHOW</td>
<td>• <strong>Adobe Analytics:</strong> a.media.show</td>
<td>• <strong>Available:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>• <strong>API Key:</strong> media.show</td>
<td>• <strong>Heartbeats:</strong> s:meta:a.media.show</td>
<td>• <strong>Reserved Variable:</strong> eVar</td>
</tr>
<tr>
<td></td>
<td>• <strong>Required:</strong> No</td>
<td>• <strong>Expiration:</strong> On HIT</td>
<td>• <strong>Report Name:</strong> Show</td>
</tr>
<tr>
<td></td>
<td>• <strong>Type:</strong> string</td>
<td>• <strong>Context Data:</strong> a.media.show</td>
<td>• <strong>Data Feed:</strong> videoshow</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sent with:</strong> Media Start, Media Close</td>
<td>• <strong>Audience Manager:</strong> c_contextdata.a.media.show</td>
<td>• <strong>Sample value:</strong> &quot;Modern Family&quot;, &quot;Blacklist&quot;, &quot;New Girl&quot;, etc.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Min. SDK Version:</strong> 1.5.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Sample value:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program/Series Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Program Name is required only if the show is part of a series.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Stream Format</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>SDK Key:</strong> STREAM_FORMAT</td>
<td>• <strong>Adobe Analytics:</strong> a.media.format</td>
<td>• <strong>Available:</strong> Use custom processing rule</td>
</tr>
<tr>
<td></td>
<td>• <strong>API Key:</strong> N/A</td>
<td>• <strong>Heartbeats:</strong> s:meta:a.media.format</td>
<td>• <strong>Reserved Variable:</strong> N/A</td>
</tr>
<tr>
<td></td>
<td>• <strong>Required:</strong> No</td>
<td></td>
<td>• <strong>Report Name:</strong> Custom</td>
</tr>
<tr>
<td></td>
<td>• <strong>Type:</strong> string</td>
<td></td>
<td>• <strong>Context Data:</strong> a.media.format</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sent with:</strong> Media Start, Media Close</td>
<td></td>
<td>• <strong>Data Feed:</strong> N/A</td>
</tr>
<tr>
<td></td>
<td>• <strong>Min. SDK Version:</strong> 1.5.7</td>
<td></td>
<td>• <strong>Audience Manager:</strong> c_contextdata.a.media.format</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sample value:</strong> &quot;Live&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Format of the stream (Live, VOD, Linear).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Season</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>SDK Key:</strong> SEASON</td>
<td>• <strong>Adobe Analytics:</strong> a.media.season</td>
<td>• <strong>Available:</strong> Yes</td>
</tr>
<tr>
<td></td>
<td>• <strong>API Key:</strong> media.season</td>
<td>• <strong>Heartbeats:</strong> s:meta:a.media.season</td>
<td>• <strong>Reserved Variable:</strong> eVar</td>
</tr>
<tr>
<td></td>
<td>• <strong>Required:</strong> No</td>
<td></td>
<td>• <strong>Expiration:</strong> On HIT</td>
</tr>
<tr>
<td></td>
<td>• <strong>Type:</strong> string</td>
<td></td>
<td>• <strong>Report Name:</strong> Season</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sent with:</strong> Media Start, Media Close</td>
<td></td>
<td>• <strong>Context Data:</strong> a.media.season</td>
</tr>
<tr>
<td></td>
<td>• <strong>Min. SDK Version:</strong> 1.5.7</td>
<td></td>
<td>• <strong>Data Feed:</strong> videoseason</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <strong>Audience Manager:</strong> c_contextdata.a.media.season</td>
</tr>
<tr>
<td>Label</td>
<td>Implementation</td>
<td>Network Parameters</td>
<td>Reporting</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>--------------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>Season</strong></td>
<td>• Sample value: &quot;2&quot; (an integer representing the season number)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>Season Series is required only if the show is part of a series.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **Episode** | • SDK Key: EPISODE  | • Adobe Analytics: | • Available: Yes |
|             | • API Key: media.episode | a.media.episode | • Reserved Variable: eVar |
|             | • Required: No | • Heartbeats: s:meta:a.media.episode | • Expiration: On HIT |
|             | • Type: string | | • Report Name: Episode |
|             | • Sent with: Media Start, Media Close | | • Context Data: a.media.episode |
|             | • Min. SDK Version: 1.5.7 | | • Data Feed: videoepisode |
|             | • Sample value: "13" (an integer representing the episode number) | | • Audience Manager: c_contextdata.a.media.episode |
|             | | | |

The number of the episode.

| **Asset ID** | • SDK Key: ASSET_ID | • Adobe Analytics: | • Available: Yes |
|             | • API Key: media.assetId | a.media.asset | • Reserved Variable: Classification |
|             | • Required: No | • Heartbeats: s:meta:a.media.asset | • Report Name: Asset ID |
|             | • Type: string | | • Context Data: a.media.asset |
|             | • Sent with: Media Start, Media Close | | • Data Feed: N/A |
|             | • Min. SDK Version: 1.5.7 | | • Audience Manager: c_contextdata.a.media.asset |
|             | • Sample value: "89745363" (any integer and/or letter combination) | | |

This is the unique identifier for the content of the media asset, such as the TV series episode identifier, movie asset identifier, or live event identifier. Typically these IDs are derived from metadata authorities such as EIDR, TMS/Gracenote, or Rovi. These identifiers can also be from other proprietary or in-house systems.

| **Genre** | • SDK Key: GENRE  | • Adobe Analytics: | • Available: Yes |
|           | • API Key: media.genre | a.media.genre | • Reserved Variable: List eVar |
|           | • Required: No | • Heartbeats: | • Expiration: On HIT |
|           | | | |
### Type
- **Type:** string
- **Sent with:** Media Start, Media Close
- **Min. SDK Version:** 1.5.7
- **Sample value:** "Drama", "Comedy", etc.

Type or grouping of content as defined by content producer. Values should be comma delimited in variable implementation. In reporting, the list eVar will split each value into a line item, with each line item receiving equal metrics weight.

### First Air Date
- **SDK Key:** FIRST_AIR_DATE
- **API Key:** media.firstAirDate
- **Required:** No
- **Type:** string
- **Sent with:** Initiate
- **Min. SDK Version:** 1.5.7
- **Sample value:** "2016-01-25"

The date when the content first aired on television. Any date format is acceptable, but Adobe recommends: YYYY-MM-DD

### First Digital Date
- **SDK Key:** FIRST_DIGITAL_DATE
- **API Key:** media.firstDigitalDate
- **Required:** No
- **Type:** string
- **Sent with:** Media Start, Media Close
- **Min. SDK Version:** 1.5.7
- **Sample value:** "2016-01-25"

The date when the content first aired on any digital channel or platform. Any date format is acceptable but Adobe recommends: YYYY-MM-DD

### Content Rating
- **SDK Key:** RATING
- **API Key:** media.rating
- **Required:** No

The date when the content first aired on any digital channel or platform. Any date format is acceptable but Adobe recommends: YYYY-MM-DD
### Reporting Network Parameters

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong>: string</td>
<td></td>
<td>s:meta:a.media.rating</td>
<td><strong>Report Name</strong>: Content Rating</td>
</tr>
<tr>
<td><strong>Sent with</strong>: Media Start, Media Close</td>
<td></td>
<td></td>
<td><strong>Context Data</strong>: a.media.rating</td>
</tr>
<tr>
<td><strong>Min. SDK Version</strong>: 1.5.7</td>
<td></td>
<td></td>
<td><strong>Data Feed</strong>: N/A</td>
</tr>
<tr>
<td><strong>Sample value</strong>: TVY, TVG, TVPG, TVMA, etc.</td>
<td></td>
<td></td>
<td><strong>Audience Manager</strong>: c_contextdata.a.media.rating</td>
</tr>
</tbody>
</table>

Rating as defined by TV Parental Guidelines.

### Originator

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SDK Key</strong>: ORIGINATOR</td>
<td></td>
<td></td>
<td><strong>Available</strong>: Yes</td>
</tr>
<tr>
<td><strong>API Key</strong>: media.originator</td>
<td></td>
<td></td>
<td><strong>Reserved Variable</strong>: Classification</td>
</tr>
<tr>
<td><strong>Required</strong>: No</td>
<td></td>
<td></td>
<td><strong>Report Name</strong>: Originator</td>
</tr>
<tr>
<td><strong>Type</strong>: string</td>
<td></td>
<td></td>
<td><strong>Context Data</strong>: a.media.originator</td>
</tr>
<tr>
<td><strong>Sent with</strong>: Media Start, Media Close</td>
<td></td>
<td></td>
<td><strong>Data Feed</strong>: N/A</td>
</tr>
<tr>
<td><strong>Min. SDK Version</strong>: 1.5.7</td>
<td></td>
<td></td>
<td><strong>Audience Manager</strong>: c_contextdata.a.media.originator</td>
</tr>
<tr>
<td><strong>Sample value</strong>: &quot;Warner Brothers&quot;, &quot;Sony&quot;, &quot;Disney&quot;, etc.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Creator of the content.

### Network

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SDK Key</strong>: NETWORK</td>
<td></td>
<td></td>
<td><strong>Available</strong>: Yes</td>
</tr>
<tr>
<td><strong>API Key</strong>: media.network</td>
<td></td>
<td></td>
<td><strong>Reserved Variable</strong>: eVar</td>
</tr>
<tr>
<td><strong>Required</strong>: No</td>
<td></td>
<td></td>
<td><strong>Expiration</strong>: On HIT</td>
</tr>
<tr>
<td><strong>Type</strong>: string</td>
<td></td>
<td></td>
<td><strong>Report Name</strong>: Network</td>
</tr>
<tr>
<td><strong>Sent with</strong>: Media Start, Media Close</td>
<td></td>
<td></td>
<td><strong>Context Data</strong>: a.media.network</td>
</tr>
<tr>
<td><strong>Min. SDK Version</strong>: 1.5.7</td>
<td></td>
<td></td>
<td><strong>Data Feed</strong>: videонetwork</td>
</tr>
<tr>
<td><strong>Sample value</strong>: &quot;Fox&quot;, &quot;Bravo&quot;, &quot;ESPN&quot;, etc.</td>
<td></td>
<td></td>
<td><strong>Audience Manager</strong>: c_contextdata.a.media.network</td>
</tr>
</tbody>
</table>

The network/channel name.

### Show Type

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SDK Key</strong>: SHOW_TYPE</td>
<td></td>
<td></td>
<td><strong>Available</strong>: Yes</td>
</tr>
<tr>
<td><strong>API Key</strong>: media.showType</td>
<td></td>
<td></td>
<td><strong>Reserved Variable</strong>: eVar</td>
</tr>
<tr>
<td><strong>Required</strong>: No</td>
<td></td>
<td></td>
<td><strong>Expiration</strong>: On HIT</td>
</tr>
<tr>
<td><strong>Type</strong>: string</td>
<td></td>
<td></td>
<td><strong>Report Name</strong>: Show Type</td>
</tr>
<tr>
<td><strong>Sent with</strong>: Media Start, Media Close</td>
<td></td>
<td></td>
<td><strong>Context Data</strong>: a.media.type</td>
</tr>
<tr>
<td><strong>Sample value</strong>:</td>
<td></td>
<td></td>
<td><strong>Data Feed</strong>: videonshowtype</td>
</tr>
</tbody>
</table>

Adobe Analytics: a.media.showType
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min. SDK Version: 1.5.7 &lt;br&gt; Sample value: &lt;br&gt; &quot;0&quot; = Full episode &lt;br&gt; &quot;1&quot; = Preview/trailer &lt;br&gt; &quot;2&quot; = Clip &lt;br&gt; &quot;3&quot; = Other</td>
<td>Audience Manager: &lt;br&gt; c_contextdata.a.media.type</td>
<td></td>
</tr>
</tbody>
</table>

Type of content, expressed as an integer between 0 and 3.

| MVPD | SDK Key: MVPD <br> API Key: media.pass.mvpd <br> Required: No <br> Type: string <br> Sent with: Media Start, Media Close <br> Min. SDK Version: 1.5.7 <br> Sample value: "Comcast", "DirecTV", "Dish", etc. | Adobe Analytics: <br> a.media.pass.mvpd <br> Heartbeats: <br> s:meta:a.media.pass.mvpd | Available: Yes <br> Reserved Variable: eVar <br> Expiration: On HIT <br> Report Name: MVPD <br> Context Data: <br> a.media.pass.mvpd <br> Data Feed: videomvpd <br> Audience Manager: <br> c_contextdata.a.media.pass.mvpd |

MVPD provided via Adobe authentication.

| Authorized | SDK Key: AUTHORIZED <br> API Key: media.pass.auth <br> Required: No <br> Type: string <br> Sent with: Media Start, Media Close <br> Min. SDK Version: 1.5.7 <br> Sample value: "TRUE" | Adobe Analytics: <br> a.media.pass.auth <br> Heartbeats: <br> s:meta:a.media.pass.auth | Available: Yes <br> Reserved Variable: event <br> Report Name: Authorized <br> Context Data: <br> a.media.pass.auth <br> Data Feed: videoauthorized <br> Audience Manager: <br> c_contextdata.a.media.pass.auth |

The user has been authorized via AdobePass.

⚠️ Important: This can only be true if it is set. If it is not set, no value is returned.

<p>| Day Part | SDK Key: DAY_PART &lt;br&gt; API Key: media.dayPart &lt;br&gt; Required: No &lt;br&gt; Type: string &lt;br&gt; Sent with: Media Start, Media Close | Adobe Analytics: &lt;br&gt; a.media.dayPart &lt;br&gt; Heartbeats: &lt;br&gt; s:meta:a.media.dayPart | Available: Yes &lt;br&gt; Reserved Variable: eVar &lt;br&gt; Expiration: On HIT &lt;br&gt; Report Name: Day Part &lt;br&gt; Context Data: &lt;br&gt; a.media.dayPart |</p>
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Min. SDK Version: 1.5.7&lt;br&gt;• Sample value:</td>
<td></td>
<td>• Data Feed: videodaypart&lt;br&gt; • Audience Manager: c_contextdata.a.media.dayPart</td>
<td>A property that defines the time of the day when the content was broadcast or played. This could have any value set as necessary by customers.</td>
</tr>
</tbody>
</table>

| Media Feed Type | SDK Key: FEED<br> • API Key: media.feed<br> • Required: No<br> • Type: string<br> • Sent with: Media Start, Media Close<br> • Min. SDK Version: 1.5.7<br> • Sample value: "East-HD", "West-HD", "East-SD", etc. | • Adobe Analytics: a.media.feed<br> • Heartbeats: s:meta:a.media.feed | Available: Yes<br> • Reserved Variable: eVar<br> • Expiration: On HIT<br> • Report Name: Media Feed Type<br> • Context Data: a.media.feed<br> • Data Feed: videofeedtype<br> • Audience Manager: c_contextdata.a.media.feed |
|                |                                                                                 |                                                                                     | Release Date: 09/13/18<br> Type of feed.                                                  |

| Artist        | SDK Key: <br> • API Key: media.artist<br> • Required: No<br> • Type: string<br> • Sent with: Media Start, Media Close<br> • Min. SDK Version: 1.5.7<br> • Sample value: "The Beatles" | • Adobe Analytics: a.media.artist<br> • Heartbeats: s:meta:a.media.artist | Available: Yes<br> • Reserved Variable: eVar<br> • Expiration: On HIT<br> • Report Name: Media Feed Type<br> • Context Data: a.media.artist<br> • Data Feed: videoaudioartist<br> • Audience Manager: c_contextdata.a.media.artist |
|               |                                                                                 |                                                                                     | Release Date: 09/13/18<br> Name of the artist.                                          |

<p>| Album         | SDK Key: &lt;br&gt; • API Key: media.album&lt;br&gt; • Required: No&lt;br&gt; • Type: string&lt;br&gt; • Sent with: Media Start, Media Close | • Adobe Analytics: a.media.album&lt;br&gt; • Heartbeats: s:meta:a.media.album | Available: Yes&lt;br&gt; • Reserved Variable: eVar&lt;br&gt; • Expiration: On HIT&lt;br&gt; • Report Name: Media Feed Type&lt;br&gt; • Context Data: a.media.album&lt;br&gt; • Data Feed: videoaudioalbum |</p>
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Min. SDK Version: 1.5.7 &lt;br&gt; • Sample value: &quot;Revolver&quot;</td>
<td>• Adobe Analytics: a.media.label &lt;br&gt; • Heartbeats: s:meta:a.media.album</td>
<td>• Audience Manager: c_contextdata.a.media.album</td>
<td></td>
</tr>
</tbody>
</table>
| **Release Date: 09/13/18**  
*Note:* Available in Media Collection API or Media SDK 2.2. | |

Name of the album.

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>• SDK Key: &lt;br&gt; • API Key: media.label &lt;br&gt; • Required: No &lt;br&gt; • Type: string &lt;br&gt; • Sent with: Media Start, Media Close &lt;br&gt; • Min. SDK Version: 1.5.7 &lt;br&gt; • Sample value: &quot;Revolver&quot;</td>
<td>• Adobe Analytics: a.media.label &lt;br&gt; • Heartbeats: s:meta:a.media.label</td>
<td>• Available: Yes &lt;br&gt; • Reserved Variable: eVar &lt;br&gt; • Expiration: On HIT &lt;br&gt; • Report Name: &lt;br&gt; • Context Data: a.media.label &lt;br&gt; • Data Feed: videoaudiolabel &lt;br&gt; • Audience Manager: c_contextdata.a.media.label</td>
<td></td>
</tr>
</tbody>
</table>
| **Release Date: 09/13/18**  
*Note:* Available in Media Collection API or Media SDK 2.2. | |

Name of the record label.

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>• SDK Key: &lt;br&gt; • API Key: media.author &lt;br&gt; • Required: No &lt;br&gt; • Type: string &lt;br&gt; • Sent with: Media Start, Media Close &lt;br&gt; • Min. SDK Version: 1.5.7 &lt;br&gt; • Sample value: &quot;John Kennedy Toole&quot;</td>
<td>• Adobe Analytics: a.media.author &lt;br&gt; • Heartbeats: s:meta:a.media.author</td>
<td>• Available: Yes &lt;br&gt; • Reserved Variable: eVar &lt;br&gt; • Expiration: On HIT &lt;br&gt; • Report Name: &lt;br&gt; • Context Data: a.media.author &lt;br&gt; • Data Feed: videoaudioauthor &lt;br&gt; • Audience Manager: c_contextdata.a.media.author</td>
<td></td>
</tr>
</tbody>
</table>
| **Release Date: 09/13/18**  
*Note:* Available in Media Collection API or Media SDK 2.2. | |

Name of the author (of an audiobook).

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>• SDK Key: &lt;br&gt; • API Key: media.station</td>
<td>• Adobe Analytics: a.media.station</td>
<td>• Available: Yes &lt;br&gt; • Reserved Variable: eVar</td>
<td></td>
</tr>
</tbody>
</table>
| **Release Date: 09/13/18**  
*Note:* Available in Media Collection API or Media SDK 2.2. | |
### Reporting Network Parameters

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Required: No</td>
<td>• Heartbeats:</td>
<td>• Expiration: On HIT</td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td>s:meta:a.media.station</td>
<td>• Report Name:</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Start, Media Close</td>
<td></td>
<td>• Context Data: a.media.station</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: 1.5.7</td>
<td></td>
<td>• Data Feed: videoaudiostream</td>
</tr>
<tr>
<td></td>
<td>• Sample value: &quot;NPR&quot;</td>
<td></td>
<td>• Audience Manager: c_contextdata.a.media.station</td>
</tr>
</tbody>
</table>

**Release Date: 09/13/18**

💡 *Note: Available in Media Collection API or Media SDK 2.2.*

Name / ID of the radio station.

### Publisher

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• SDK Key:</td>
<td>• Adobe Analytics:</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: media.publisher</td>
<td>a.media.publisher</td>
<td>• Reserved Variable: eVar</td>
</tr>
<tr>
<td></td>
<td>• Required: No</td>
<td>• Heartbeats:</td>
<td>• Expiration: On HIT</td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td>s:meta:a.media.publisher</td>
<td>• Report Name:</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Start, Media Close</td>
<td></td>
<td>• Context Data: a.media.publisher</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: 1.5.7</td>
<td></td>
<td>• Data Feed: videoaudiopublisher</td>
</tr>
<tr>
<td></td>
<td>• Sample value: &quot;Random Bauhaus&quot;</td>
<td></td>
<td>• Audience Manager: c_contextdata.a.media.publisher</td>
</tr>
</tbody>
</table>

**Release Date: 09/13/18**

💡 *Note: Available in Media Collection API or Media SDK 2.2.*

Name of the audio content publisher.

### Audio and Video Metrics

#### Media Starts

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Starts</td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics:</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: N/A</td>
<td>a.media.view</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td>• Heartbeats:</td>
<td>• Report Name: Media Starts</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Initiate</td>
<td>s:event:type=start</td>
<td>• Context Data: a.media.view</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td>• Data Feed: videostart</td>
</tr>
<tr>
<td></td>
<td>• Sample value: TRUE</td>
<td></td>
<td>• Audience Manager: c_contextdata.a.media.view</td>
</tr>
</tbody>
</table>
## Reporting Network Parameters

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Release Date:</strong> 09/13/18</td>
<td>Load event for the media. (This occurs when the viewer clicks the Play button). This would count even if there are pre-roll ads, buffering, errors, and so on.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Important:</strong> This can only be true if it is set. If it is not set, no value is returned.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Content Starts</strong></td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: N/A</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: N/A</td>
<td>• Heartbeats: N/A</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td></td>
<td>• Report Name: Content Starts</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td></td>
<td>• Context Data: a.media.play</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td>• Data Feed: videoplay</td>
</tr>
<tr>
<td></td>
<td>• Sample value: TRUE</td>
<td></td>
<td>• Audience Manager: c_contextdata.a.media.play</td>
</tr>
<tr>
<td></td>
<td>First frame of media is consumed. If the user drops during ad, buffering, etc., then there would be no &quot;Content Start&quot; event.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Important:</strong> This can only be true if it is set. If it is not set, no value is returned.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Content Complete</strong></td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: N/A</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: N/A</td>
<td>• Heartbeats:</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td>s:event:type=complete</td>
<td>• Report Name: Content Completes</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td></td>
<td>• Context Data: a.media.complete</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td>• Data Feed: videocomplete</td>
</tr>
<tr>
<td></td>
<td>• Sample value: TRUE</td>
<td></td>
<td>• Audience Manager: c_contextdata.a.media.complete</td>
</tr>
<tr>
<td></td>
<td>A stream that was watched to completion - This does not necessarily mean the user watched or listened to the whole stream; they could have skipped ahead. This only means the user reached the end of the stream, 100%.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Important:</strong> This can only be true if it is set. If it is not set, no value is returned.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Content Time Spent</strong></td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: N/A</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: N/A</td>
<td>• Heartbeats: N/A</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td></td>
<td>• Report Name: Content Time Spent</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td></td>
<td>• Context Data:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a.media.time_spent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Data Feed: videotime_spent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Audience Manager: c_contextdata.a.media.time_spent</td>
</tr>
<tr>
<td>Label</td>
<td>Implementation</td>
<td>Network Parameters</td>
<td>Reporting</td>
</tr>
<tr>
<td>-------</td>
<td>----------------</td>
<td>--------------------</td>
<td>-----------</td>
</tr>
</tbody>
</table>
| • Min. SDK Version: Any  
• Sample value: 105 | | | • Context Data:  
a.media.timePlayed  
• Data Feed: videotime  
• Audience Manager:  
c_contextdata.a.media.timePlayed |

**Release Date: 9/13/18**

Sums the event duration (in seconds) for all events of type PLAY on the main content.

The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports & Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds.

| Media Time Spent | | | |
| • SDK Key: Automatically set  
• API Key: N/A  
• Type: number  
• Sent with: Media Close  
• Min. SDK Version: Any  
• Sample value: 120 | • Adobe Analytics: N/A  
• Heartbeats: N/A | • Available: Yes  
• Reserved Variable: event  
• Report Name: Media Time Spent  
• Context Data:  
a.media.totalTimePlayed  
• Data Feed: videototaltime  
• Audience Manager:  
c_contextdata.a.media.totalTimePlayed |

**Release Date: 09/13/18**

Sums the event duration (in seconds) for all events of type PLAY, both main and ad content.

The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports & Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds.

| Unique Time Played | | | |
| • SDK Key: Automatically set  
• API Key: N/A  
• Type: number  
• Sent with: Media Close  
• Min. SDK Version: Any  
• Sample value: 94 | • Adobe Analytics: N/A  
• Heartbeats: N/A | • Available: Yes  
• Reserved Variable: event  
• Report Name: Custom  
• Context Data:  
a.media.uniqueTimePlayed  
• Data Feed: videouniquetimeplayed  
• Audience Manager:  
c_contextdata.a.media.uniqueTimePlayed |

**Release Date: 09/13/18**

The value in seconds of the unique segments of content played during a session. Excludes time played on seek back scenarios in which a viewer is watching the same segment of the content multiple times.
The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports & Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds.

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>10% Progress Marker</td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: N/A</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: N/A</td>
<td>• Heartbeats: N/A</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td></td>
<td>• Report Name: 10% Progress Marker</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td></td>
<td>• Context Data:</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td>a.media.progress10</td>
</tr>
<tr>
<td></td>
<td>• Sample value: TRUE</td>
<td></td>
<td>• Data Feed: videoprogress10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Audience Manager:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c_contextdata.a.media.progress10</td>
</tr>
</tbody>
</table>

Playhead passes the 10% marker of content based on length. The marker is only counted once, even if seeking backwards. If seeking forward, markers that are skipped are not counted.

⚠️ **Important:** This can only be true if it is set. If it is not set, no value is returned.

| 25% Progress Marker  | • SDK Key: Automatically set | • Adobe Analytics: N/A | • Available: Yes |
|----------------------| • API Key: N/A            | • Heartbeats: N/A   | • Reserved Variable: event        |
|                      | • Type: string            |                     | • Report Name: 25% Progress Marker|
|                      | • Sent with: Media Close  |                     | • Context Data:                   |
|                      | • Min. SDK Version: Any   |                     | a.media.progress25                |
|                      | • Sample value: TRUE      |                     | • Data Feed: videoprogress25      |
|                      |                            |                     | • Audience Manager:               |
|                      |                            |                     | c_contextdata.a.media.progress25  |

Playhead passes the 25% marker of content based on content length. Marker only counted once, even if seeking backwards. If seeking forward, markers that are skipped are not counted.

⚠️ **Important:** This can only be true if it is set. If it is not set, no value is returned.

<p>| 50% Progress Marker  | • SDK Key: Automatically set | • Adobe Analytics: N/A | • Available: Yes |
|----------------------| • API Key: N/A            | • Heartbeats: N/A   | • Reserved Variable: event        |
|                      | • Type: string            |                     | • Report Name: 50% Progress Marker|
|                      | • Sent with: Media Close  |                     | • Context Data:                   |
|                      | • Min. SDK Version: Any   |                     | a.media.progress50                |
|                      | • Sample value: TRUE      |                     | • Data Feed: videoprogress50      |</p>
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Audience Manager:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c_contextdata.a.media.progress50</td>
</tr>
</tbody>
</table>

Playhead passes the 50% marker of content based on content length. Marker only counted once, even if seeking backwards. If seeking forward, markers that are skipped are not counted.

⚠️ **Important:** This can only be true if it is set. If it is not set, no value is returned.

| 75% Progress Marker | • SDK Key: Automatically set | • Adobe Analytics: N/A | • Available: Yes |
|                     | • API Key: N/A              | • Heartbeats: N/A      | • Reserved Variable: event |
|                     | • Type: string              |                     | • Report Name: 75% Progress Marker |
|                     | • Sent with: Media Close    |                     | • Context Data: |
|                     | • Min. SDK Version: Any    |                     | a.media.progress75 |
|                     | • Sample value: TRUE       |                     | • Data Feed: videoprogress75 |
|                     |                          |                     | • Audience Manager: |
|                     |                          |                     | c_contextdata.a.media.progress75 |

Playhead passes the 75% marker of content based on content length. Marker only counted once, even if seeking backwards. If seeking forward, markers that are skipped are not counted.

⚠️ **Important:** This can only be true if it is set. If it is not set, no value is returned.

| 95% Progress Marker | • SDK Key: Automatically set | • Adobe Analytics: N/A | • Available: Yes |
|                     | • API Key: N/A              | • Heartbeats: N/A      | • Reserved Variable: event |
|                     | • Type: string              |                     | • Report Name: 95% Progress Marker |
|                     | • Sent with: Media Close    |                     | • Context Data: |
|                     | • Min. SDK Version: Any    |                     | a.media.progress95 |
|                     | • Sample value: TRUE       |                     | • Data Feed: videoprogress95 |
|                     |                          |                     | • Audience Manager: |
|                     |                          |                     | c_contextdata.a.media.progress95 |

Playhead passes the 95% marker of content based on content length. Marker only counted once, even if seeking backwards. If seeking forward, markers that are skipped are not counted.

⚠️ **Important:** This can only be true if it is set. If it is not set, no value is returned.
### Average Minute Audience

- **Label:** Average Minute Audience
- **Implementation:**
  - **SDK Key:** Automatically set
  - **API Key:** N/A
  - **Type:** number
  - **Sent with:** Media Close
  - **Min. SDK Version:** Any
  - **Sample value:** Greater than or equal to 1
- **Network Parameters:**
  - **Adobe Analytics:** N/A
  - **Heartbeats:** N/A
- **Reporting:**
  - **Available:** Yes
  - **Reserved Variable:** event
  - **Report Name:** Average Minute Audience
  - **Context Data:** a.media.averageMinuteAudience
  - **Data Feed:** videoaverage_minute_audience
  - **Audience Manager:** c_contextdata.a.media.averageMinuteAudience

**Average Minute Audience** metric is computed as Total Content Time Spent, for one specific media item, divided by its length for all of its playback sessions:

\[
\text{average\_minute\_audience} = \frac{\text{timeSpent}}{\text{videoLength}};
\]

### Estimated Streams

- **Label:** Estimated Streams
- **Implementation:**
  - **SDK Key:** Automatically set
  - **API Key:** N/A
  - **Type:** number
  - **Sent with:** Media Close
  - **Min. SDK Version:** Any
  - **Sample value:**
    - 1 - For a 19 minutes playback
    - 2 - For a 31 minutes playback
    - 3 - For a 78 minutes playback
- **Network Parameters:**
  - **Adobe Analytics:** N/A
  - **Heartbeats:** N/A
- **Reporting:**
  - **Available:** Use custom processing rule
  - **Reserved Variable:** N/A
  - **Report Name:** Custom
  - **Context Data:** a.media.estimatedStreams
  - **Data Feed:** N/A
  - **Audience Manager:** c_contextdata.a.media.estimatedStreams

The estimated number of video or audio streams per each individual content. This value is increased for each 30 minutes of play time (content + ads). Customers must create their own processing rules to have the value available for reporting.

A stream is counted at every 30 minutes, based on the `ms_s` or `totalTimePlayed = Video Total Time`, similar to:

\[
\text{estimatedStreams} = \text{FLOOR}(\frac{\text{ms\_s}}{1800}) + 1
\]

### Paused Impacted Streams

- **Label:** Paused Impacted Streams
- **Implementation:**
  - **SDK Key:** Automatically set
  - **API Key:** N/A
  - **Type:** string
  - **Sent with:** Media Close
  - **Min. SDK Version:** 1.5.6
  - **Sample value:** TRUE
- **Network Parameters:**
  - **Adobe Analytics:** N/A
  - **Heartbeats:**
    - s:event:type=pause
- **Reporting:**
  - **Available:** Yes
  - **Reserved Variable:** event
  - **Report Name:** Paused Impacted Stream
  - **Context Data:** a.media.pause
  - **Data Feed:** videopause
  - **Audience Manager:** c_contextdata.a.media.pause

Paused Impacted Streams metric represents the number of times a stream was paused and then resumed. This value increases at each 30 minutes for every paused stream (content + ads). Customers must create their own processing rules to have the value available for reporting.
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pause Events</strong></td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: N/A</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: N/A</td>
<td>• Heartbeats:</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td>s:event:type=pause</td>
<td>• Report Name: Pause Events</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td></td>
<td>• Context Data:</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: 1.5.6</td>
<td>a.media.pauseCount</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sample value: 2 (Integer)</td>
<td></td>
<td>• Data Feed: videopausecount</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Audience Manager:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c_contextdata.a.media.pauseCount</td>
</tr>
<tr>
<td><strong>Total Pause Duration</strong></td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: N/A</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: N/A</td>
<td>• Heartbeats: N/A</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td></td>
<td>• Report Name: Total Pause Duration</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td>• Context Data:</td>
<td>• Data Feed: videopausetime</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: 1.5.6</td>
<td>a.media.pauseTime</td>
<td>• Audience Manager:</td>
</tr>
<tr>
<td></td>
<td>• Sample value: 190</td>
<td></td>
<td>c_contextdata.a.media.pauseTime</td>
</tr>
</tbody>
</table>

This metric is computed as a count of pause periods that occurred during a playback session.

**Release Date: 09/13/18**

Sums the duration (in seconds) of all events of type PAUSE.

The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports & Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds.

<table>
<thead>
<tr>
<th>Content Resumes</th>
<th>• SDK Key: Automatically set</th>
<th>• Adobe Analytics: N/A</th>
<th>• Available: Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• API Key: media.resume</td>
<td>• Heartbeats:</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td>s:event:type=resume</td>
<td>• Report Name: Content Resumes</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td></td>
<td>• Context Data: a.media.resume</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: 1.5.6</td>
<td></td>
<td>• Data Feed: videoresume</td>
</tr>
<tr>
<td></td>
<td>• Sample value: TRUE</td>
<td></td>
<td>• Audience Manager:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c_contextdata.a.media.resume</td>
</tr>
</tbody>
</table>
### Reporting Network Parameters

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Resume is counted for each playback that resumes after more than 30 minutes of buffer, pause, or stall period OR if this value is set by the player on the VideoInfo object before trackPlay.</td>
<td><img src="image" alt="Important" /> This can only be true if it is set. If it is not set, no value is returned.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Content Segment Views
- **SDK Key**: Automatically set
- **API Key**: N/A
- **Type**: string
- **Sent with**: Media Close
- **Min. SDK Version**: Any
- **Sample value**: TRUE

| • Adobe Analytics: N/A | • Available: Yes | • Reserved Variable: event |
| • Heartbeats: N/A | • Report Name: Content Segment Views | |
| • Context Data: a.media.segmentView | • Data Feed: videosegmentviews | • Audience Manager: c_contextdata.a.media.segmentView |

The number of views of the main content. A Content Segment View is counted when there is at least one frame viewed.

![Important](image) This can only be true if it is set. If it is not set, no value is returned.

### Ad parameters

This topic presents a list of video ad data, including context data values, that Adobe collects via solution variables.

This section contains the following information:

- **Ad Video Data**
- **Standard Ad Metadata**
- **Ad Metrics**

Table data description:

- **Label**: The name of the parameter.
- **Implementation**: Information on implementation values and requirements
- **Key**: Variable, set either manually in your app, or automatically by the Adobe Media SDK.
- **Required**: Indicates whether the parameter is required for basic video tracking.
- **Type**: Specifies the type of the variable to be set, string or number.
- **Sent With**: Indicates when the data is sent: *Initiate* is the analytics call sent on video start, *Ad Start* is the analytics call sent on ad start, *Chapter Start* is the analytics call sent on chapter start, and *Close* is the compiled analytics call sent directly from the heartbeat server to the analytics server at the end of the media session, or the end of the ad. The Close calls are not available in network packet calls.
- **Min. SDK Version**: Indicates which SDK version you would need to access the parameter.
• **Sample Value** - Provides example of common variable usage.

• **Network Parameters**: Displays the values that are passed to Adobe Analytics or Heartbeat servers. This column shows the names of the parameters that are seen in the network calls generated by Adobe Media SDKs.

• **Reporting**: Information on how to view and analyze the video data.

  • **Available** - Indicates whether the data is available in reporting by default (Yes), or requires custom set-up (Custom)
  • **Reserved Variable** - Indicates whether the data is captured as an event, eVar, prop, or classification in a reserved variable.
  • **Report Name** - Name of Adobe Analytics report for variable
  • **Context Data** - Name of the Adobe Analytics context data passed to the reporting server and used in processing rules.
  • **Data Feed** - Column name for variable found in Clickstream or Live Stream data feeds
  • **Audience Manager** - Trait name found in Adobe Audience Manager

**Important**: Do not change the classification names for any variables listed below that are described under Reporting/Reserved Variable as “classification”.

The media classifications are defined when a report suite is enabled for media tracking. From time to time, Adobe adds new properties, and, when this occurs, customers must re-enable their report suites to get access to the new media properties. During the update process Adobe determines whether the classifications are enabled by checking the names of the variables. If any of them is missing, Adobe adds the missing ones again.

### Ad Video Data

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
</table>
| **Ad ID**              | • **SDK Key**: adId*  
• **API Key**: media.ad.id  
• **Required**: Yes  
• **Type**: string  
• **Sent with**: Ad Start, Ad Close  
• **Min. SDK Version**: Any  
• **Sample value**: “2125” | • **Adobe Analytics**:  
  a.media.ad.name  
  
• **Heartbeat**:  
  s:asset:ad_id | • **Available**: Yes  
• **Reserved Variable**: eVar  
• **Expiration**: On VISIT  
• **Report Name**: Ad  
• **Context Data**:  
  a.media.ad.name  
  
• **Data Feed**: videoad  
• **Audience Manager**:  
  c_contextdata.a.media.ad.name |
| **Ad In Pod Position** | • **SDK Key**: position*  
• **API Key**: media.ad.podPosition  
• **Required**: Yes  
• **Type**: number | • **Adobe Analytics**:  
  a.media.ad.podPosition  
  
• **Heartbeat**:  
  s:asset:pod_position | • **Available**: Yes  
• **Reserved Variable**: eVar  
• **Expiration**: On HIT  
• **Report Name**: Ad In Pod Position |

ID of the ad. (Any integer and/or letter combination)

*createAdObject*(name, adId, position, length)
### Reporting Network Parameters

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
</table>
| • **Sent with:** Ad Start, Ad Close  
  • **Min. SDK Version:** Any  
  • **Sample value:** 1 | • **Context Data:** a.media.ad.podPosition  
  • **Data Feed:** videoadinpod  
  • **Audience Manager:** c_contextdata.a.media.ad.podPosition | | |
| The position (index) of the ad inside the parent ad break. The first ad has index 0, the second ad has index 1, etc. | | | |
|  
  * `createAdObject(name, adId, position, length)` | | | |

#### Ad Length

| Ad Length | • **SDK Key:** `length`*  
  • **API Key:** media.ad.length  
  • **Required:** Yes  
  • **Type:** number  
  • **Sent with:** Ad Start, Ad Close  
  • **Min. SDK Version:** 1.5.1  
  • **Sample value:** "15" | • **Adobe Analytics:** a.media.ad.length  
  • **Heartbeat:** 1:asset:ad_length | • **Available:** Yes  
  • **Reserved Variable:** eVar and classification  
  • **Expiration:** On HIT  
  • **Report Name:** Ad Length and Ad Length (variable)  
  • **Context Data:** a.media.ad.length  
  • **Data Feed:** videoadlength  
  • **Audience Manager:** c_contextdata.a.media.ad.length |
| Length of video ad in seconds. | | | |
|  
  * `createAdObject(name, adId, position, length)` | | | |

#### Ad Player Name

| Ad Player Name | • **SDK Key:** `playerName`*  
  • **API Key:** media.ad.playerName  
  • **Required:** Yes  
  • **Type:** string  
  • **Sent with:** Ad Start, Ad Close  
  • **Min. SDK Version:** Any  
  • **Sample value:** "Freewheel", etc. | • **Adobe Analytics:** a.media.ad.playerName  
  • **Heartbeat:** s:sp:player_name | • **Available:** Yes  
  • **Reserved Variable:** eVar  
  • **Expiration:** On HIT  
  • **Report Name:** Ad Player Name  
  • **Context Data:** a.media.ad.playerName  
  • **Data Feed:** videoadplayername  
  • **Audience Manager:** c_contextdata.a.media.ad.playerName |
| The name of the player responsible for rendering the ad. | | | |
|  
  * `MediaHeartbeatConfig.playerName` | | | |

#### Ad Break Name

<p>| Ad Break Name | • <strong>SDK Key:</strong> <code>name</code>* | • <strong>Adobe Analytics:</strong> a.media.ad.podFriendlyName | • <strong>Available:</strong> Yes |
| | | | |</p>
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• API Key: <code>media.ad.podFriendlyName</code></td>
<td>• Heartbeat: <code>s:asset:pod_name</code></td>
<td>• Reserved Variable: Classification</td>
</tr>
<tr>
<td></td>
<td>• Required:</td>
<td></td>
<td>• Report Name: <code>Pod Name</code></td>
</tr>
<tr>
<td></td>
<td>• SDK: Yes</td>
<td></td>
<td>• Context Data: <code>a.media.ad.podFriendlyName</code></td>
</tr>
<tr>
<td></td>
<td>• API: No</td>
<td></td>
<td>• Data Feed: <code>videoadpod</code></td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td></td>
<td>• Audience Manager: <code>c_contextdata.a.media.ad.podFriendlyName</code></td>
</tr>
<tr>
<td></td>
<td>• Sent with: Ad Start, Ad Close</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sample value: &quot;pre-roll&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ad Break Index</strong></td>
<td>• SDK Key: <code>position</code>*</td>
<td>• Adobe Analytics: <code>media.ad.podPosition</code></td>
<td>• Available: No</td>
</tr>
<tr>
<td></td>
<td>• API Key: <code>media.ad.podPosition</code></td>
<td>• Heartbeat: <code>l:asset:pod_offset</code></td>
<td>• Reserved Variable: N/A</td>
</tr>
<tr>
<td></td>
<td>• Required: Yes</td>
<td></td>
<td>• Report Name: <code>N/A</code></td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td></td>
<td>• Context Data: <code>N/A</code></td>
</tr>
<tr>
<td></td>
<td>• Sent with:</td>
<td></td>
<td>• Data Feed: <code>Pod Position</code></td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td>• Audience Manager: <code>Pod Position</code></td>
</tr>
<tr>
<td></td>
<td>• Sample value: 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ad Break Position</strong></td>
<td>• SDK Key: <code>startTime</code>*</td>
<td>• Adobe Analytics: <code>a.media.ad.podSecond</code></td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: <code>media.ad.podSecond</code></td>
<td>• Heartbeat: <code>1:asset:pod_offset</code></td>
<td>• Reserved Variable: Classification</td>
</tr>
<tr>
<td></td>
<td>• Required: Yes</td>
<td></td>
<td>• Report Name: <code>Pod Position</code></td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td></td>
<td>• Context Data: <code>Pod Position</code></td>
</tr>
<tr>
<td></td>
<td>• Sent with: Ad Start, Ad Close</td>
<td></td>
<td>• Data Feed: <code>Pod Position</code></td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td>• Audience Manager: <code>Pod Position</code></td>
</tr>
<tr>
<td></td>
<td>• Sample value: 90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Metrics and metadata</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Ad Break ID
- **SDK Key**: Automatically set
- **API Key**: N/A
- **Required**: Yes
- **Type**: string
- **Sent with**: Ad Start, Ad Close
- **Min. SDK Version**: Any
- **Sample value**: c4a577424c84067899b807c76722d495

### Network Parameters
- **Adobe Analytics**: a.media.ad.pod
- **Heartbeat**: l:asset:pod_id

### Reporting
- **Available**: Yes
- **Reserved Variable**: eVar
- **Expiration**: On HIT
- **Report Name**: Ad Pod
- **Context Data**: a.media.ad.pod
- **Data Feed**: videoadpod
- **Audience Manager**:

### Friendly name of the ad.
In reporting, "Ad Name" is the classification and "Ad Name (variable)" is the eVar.

*createAdObject(name, adId, position, length)*

## Ad Name
- **SDK Key**: name*
- **API Key**: media.ad.name
- **Required**: No
- **Type**: string
- **Sent with**: Ad Start, Ad Close
- **Min. SDK Version**: 1.5.1
- **Sample value**: “Ford F-150”

### Network Parameters
- **Adobe Analytics**: a.media.ad.friendlyName
- **Heartbeat**: s:asset:ad_name

### Reporting
- **Available**: Yes
- **Reserved Variable**: eVar and classification
- **Expiration**: On HIT
- **Report Name**: Ad Name and Ad Name (variable)
- **Context Data**: a.media.ad.friendlyName
- **Data Feed**: N/A
- **Audience Manager**:

c_contextdata.a.media.ad.friendlyName

## Standard Ad Metadata

### Advertiser
- **SDK Key**: ADVERTISER
- **API Key**: media.ad.advertiser
- **Required**: No
- **Type**: string
- **Sent with**: Ad Start, Ad Close
- **Min. SDK Version**: 1.5.7
- **Sample value**:

### Network Parameters
- **Adobe Analytics**: a.media.ad.advertiser
- **Heartbeat**: s:meta:a.media.ad.advertiser

### Reporting
- **Available**: Yes
- **Reserved Variable**: eVar
- **Expiration**: On HIT
- **Report Name**: Advertiser
- **Context Data**: Advertiser
- **Data Feed**: videoadvertiser
- **Audience Manager**:

c_contextdata.a.media.ad.advertiser
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company/Brand whose product is featured in the ad.</td>
<td>MediaHeartbeat.AdMetadataKeys</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Campaign ID** | • SDK Key: CAMPAIGN_ID  
• API Key:  
  `media.ad.campaignId`  
• Required: No  
• Type: string  
• Sent with: Ad Start, Ad Close  
• Min. SDK Version: 1.5.7  
• Sample value: Integer, or name (string). | • Adobe Analytics:  
  `a.media.ad.campaign`  
• Heartbeat:  
  `s:meta:a.media.ad.campaign` | • Available: Yes  
• Reserved Variable: eVar  
• Expiration: On HIT  
• Report Name: Campaign ID  
• Context Data:  
  `a.media.ad.campaign`  
• Data Feed: videocampaign  
• Audience Manager:  
  `c_contextdata.a.media.ad.campaign` |
| **Creative ID** | • SDK Key: CREATIVE_ID  
• API Key:  
  `media.ad.creativeId`  
• Required: No  
• Type: string  
• Sent with: Ad Start, Ad Close  
• Min. SDK Version: 1.5.7  
• Sample value: Integer, or name (string). | • Adobe Analytics:  
  `a.media.ad.creative`  
• Heartbeat:  
  `s:meta:a.media.ad.creative` | • Available: Yes  
• Reserved Variable: eVar  
• Expiration: On HIT  
• Report Name: Creative ID  
• Context Data:  
  `a.media.ad.creative`  
• Data Feed:  
  `adclassificationcreative`  
• Audience Manager:  
  `c_contextdata.a.media.ad.creative` |
| **Site ID**   | • SDK Key: SITE_ID  
• API Key:  
  `media.ad.siteId`  
• Required: No  
• Type: string  
• Sent with: Ad Start, Ad Close  
• Min. SDK Version: 1.5.7  
• Sample value: | • Adobe Analytics:  
  `a.media.ad.site`  
• Heartbeat:  
  `s:meta:a.media.ad.site` | • Available: Use custom processing rule  
• Reserved Variable: eVar  
• Expiration: On HIT  
• Report Name:  
• Context Data:  
  `a.media.ad.site`  
• Data Feed: N/A  
• Audience Manager:  
  `c_contextdata.a.media.ad.site` |
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID of the ad site.</td>
<td>MediaHeartbeat.AdMetadataKeys</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creative URL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• SDK Key: CREATIVE_URL</td>
<td></td>
<td>• Adobe Analytics:</td>
<td>• Available: Use custom processing rule</td>
</tr>
<tr>
<td>• API Key: media.ad.creativeURL</td>
<td></td>
<td>a.media.ad.creativeURL</td>
<td></td>
</tr>
<tr>
<td>• Required: No</td>
<td></td>
<td>• Heartbeat:</td>
<td>• Reserved Variable: eVar</td>
</tr>
<tr>
<td>• Type: string</td>
<td></td>
<td>s:meta:a.media.ad.creativeURL</td>
<td></td>
</tr>
<tr>
<td>• Sent with: Ad Start, Ad Close</td>
<td></td>
<td>• Expiration: On HIT</td>
<td></td>
</tr>
<tr>
<td>• Min. SDK Version: 1.5.7</td>
<td></td>
<td>• Report Name:</td>
<td></td>
</tr>
<tr>
<td>• Sample value:</td>
<td></td>
<td>• Context Data:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>a.media.ad.creativeURL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Data Feed: N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Audience Manager:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c_contextdata.a.media.ad.creativeURL</td>
<td></td>
</tr>
<tr>
<td>URL of the ad creative.</td>
<td>MediaHeartbeat.AdMetadataKeys</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Placement ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• SDK Key: PLACEMENT_ID</td>
<td></td>
<td>• Adobe Analytics:</td>
<td>• Available: Use custom processing rule</td>
</tr>
<tr>
<td>• API Key: media.ad.placementId</td>
<td></td>
<td>a.media.ad.placement</td>
<td></td>
</tr>
<tr>
<td>• Required: No</td>
<td></td>
<td>• Heartbeat:</td>
<td>• Reserved Variable: eVar</td>
</tr>
<tr>
<td>• Type: string</td>
<td></td>
<td>s:meta:a.media.ad.placement</td>
<td></td>
</tr>
<tr>
<td>• Sent with: Ad Start, Ad Close</td>
<td></td>
<td>• Expiration: On HIT</td>
<td></td>
</tr>
<tr>
<td>• Min. SDK Version: 1.5.7</td>
<td></td>
<td>• Report Name:</td>
<td></td>
</tr>
<tr>
<td>• Sample value:</td>
<td></td>
<td>• Context Data:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>a.media.ad.placement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Data Feed: N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Audience Manager:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>c_contextdata.a.media.ad.placement</td>
<td></td>
</tr>
<tr>
<td>Placement ID of the ad.</td>
<td>MediaHeartbeat.AdMetadataKeys</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ad Metrics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ad Start</td>
<td></td>
<td>• Adobe Analytics:</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td>• SDK Key: Automatically set</td>
<td></td>
<td>a.media.ad.view</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td>• API Key: N/A</td>
<td></td>
<td>• Heartbeat:</td>
<td>• Report Name: Ad Starts</td>
</tr>
<tr>
<td>• Required: Yes</td>
<td></td>
<td>s:event:type=start</td>
<td>• Data Feed: videoadstart</td>
</tr>
<tr>
<td>• Type: string</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Chapter parameters

This topic presents a list of chapter and/or segment data, including context data values, that Adobe collects via solution variables.

This section contains the following information:

- Chapter Metadata
- Chapter Metrics
Table data description:

- **Label**: The name of the parameter.
- **Implementation**: Information on implementation values and requirements
  - **Key**: Variable, set either manually in your app, or automatically by the Adobe Media SDK.
  - **Required**: Indicates whether the parameter is required for basic video tracking.
  - **Type**: Specifies the type of the variable to be set, string or number.
  - **Sent With**: Indicates when the data is sent: *Initiate* is the analytics call sent on video start, *Ad Start* is the analytics call sent on ad start, *Chapter Start* is the analytics call sent on chapter start, and *Close* is the compiled analytics call sent directly from the heartbeat server to the analytics server at the end of the media session, or the end of the ad. The Close calls are not available in network packet calls.
  - **Min. SDK Version**: Indicates which SDK version you would need to access the parameter.
  - **Sample Value**: Provides example of common variable usage.
- **Network Parameters**: Displays the values that are passed to Adobe Analytics or Heartbeat servers. This column shows the names of the parameters that are seen in the network calls generated by Adobe Media SDKs.
- **Reporting**: Information on how to view and analyze the video data.
  - **Available**: Indicates whether the data is available in reporting by default (Yes), or requires custom set-up (Custom)
  - **Reserved Variable**: Indicates whether the data is captured as an event, eVar, prop, or classification in a reserved variable.
  - **Report Name**: Name of Adobe Analytics report for variable
  - **Context Data**: Name of the Adobe Analytics context data passed to the reporting server and used in processing rules.
  - **Data Feed**: Column name for variable found in Clickstream or Live Stream data feeds
  - **Audience Manager**: Trait name found in Adobe Audience Manager

⚠️ **Important**: Do not change the classification names for any variables listed below that are described under **Reporting/Reserved Variable** as “classification”.

The media classifications are defined when a report suite is enabled for media tracking. From time to time, Adobe adds new properties, and, when this occurs, customers must re-enable their report suites to get access to the new media properties. During the update process Adobe determines whether the classifications are enabled by checking the names of the variables. If any of them is missing, Adobe adds the missing ones again.

### Chapter Metadata

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chapter Name</strong></td>
<td><strong>SDK Key</strong>: <em>name</em></td>
<td><strong>Adobe Analytics</strong>:</td>
<td><strong>Available</strong>: Created by default...</td>
</tr>
<tr>
<td></td>
<td><strong>API Key</strong>: <em>media.chapter.friendlyName</em></td>
<td><strong>a.media.chapter.friendlyName</strong></td>
<td><strong>Reserved Variable</strong>: Classification</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong>: No</td>
<td><strong>Heartbeat</strong>:</td>
<td><strong>Report Name</strong>: Chapter Name</td>
</tr>
<tr>
<td></td>
<td><strong>Type</strong>: string</td>
<td><strong>s:stream:chapter_name</strong></td>
<td><strong>Context Data</strong>:</td>
</tr>
<tr>
<td></td>
<td><strong>Sent with</strong>: Chapter Start, Chapter Close</td>
<td></td>
<td><strong>Data Feed</strong>: N/A</td>
</tr>
<tr>
<td>Label</td>
<td>Implementation</td>
<td>Network Parameters</td>
<td>Reporting</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>--------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Chapter Position</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SDK Key: <em>position</em></td>
<td>• Adobe Analytics: a.media.chapter.position</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: media.chapter.index</td>
<td>• Heartbeat: l:stream:chapter_pos</td>
<td>• Reserved Variable: Classification</td>
</tr>
<tr>
<td></td>
<td>• Required:</td>
<td>• Report Name: Chapter Position</td>
<td>• Context Data: a.media.chapter.position</td>
</tr>
<tr>
<td></td>
<td>• SDK: No</td>
<td>• Data Feed:</td>
<td>• Audience Manager: c.contextdata.a.media.chapter.position</td>
</tr>
<tr>
<td></td>
<td>• API: Yes</td>
<td>• Audience Manager:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sent with: Chapter Close</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sample value: 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The position (index, integer) of the chapter inside the content.

* `createChapterObject(name, position, length, startTime)`

<table>
<thead>
<tr>
<th>Chapter Offset</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• SDK Key: <em>startTime</em></td>
<td>• Adobe Analytics: a.media.chapter.offset</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: media.chapter.offset</td>
<td>• Heartbeat: l:stream:chapter_offset</td>
<td>• Reserved Variable: Classification</td>
</tr>
<tr>
<td></td>
<td>• Required:</td>
<td>• Report Name: Chapter Offset</td>
<td>• Context Data: a.media.chapter.offset</td>
</tr>
<tr>
<td></td>
<td>• SDK: No</td>
<td>• Data Feed:</td>
<td>• Audience Manager: c.contextdata.a.media.chapter.offset</td>
</tr>
<tr>
<td></td>
<td>• API: Yes</td>
<td>• Audience Manager:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sent with: Chapter Close</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: 1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sample value: 58</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The offset of the chapter inside the content (in seconds) from the start.

* `createChapterObject(name, position, length, startTime)`

<table>
<thead>
<tr>
<th>Chapter Length</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• SDK Key: media.chapter.length</td>
<td>• Adobe Analytics: a.media.chapter.length</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key:</td>
<td>• Heartbeat:</td>
<td>• Reserved Variable: Classification</td>
</tr>
</tbody>
</table>

The name of the chapter and/or segment.

* `createChapterObject(name, position, length, startTime)`
### l:stream:chapter_length

- **Required:**
- **SDK:** No
- **API:** Yes
- **Type:** number
- **Sent with:** Chapter Close
- **Min. SDK Version:** 1.3
- **Sample value:** 486

**Network Parameters:**
- `l:stream:chapter_length`

**Reporting:**
- **Report Name:** Chapter Length
- **Context Data:** `a.media.chapter.length`
- **Data Feed:**
- **Audience Manager:** `c_contextdata.a.media.chapter.length`

The length of the chapter, in seconds.

* `createChapterObject(name, position, length, startTime)`

### Chapter

- **SDK Key:** Automatically set
- **API Key:** N/A
- **Required:** No
- **Type:** string
- **Sent with:** Chapter Close
- **Min. SDK Version:** 1.3
- **Sample value:**

**Adobe Analytics:**
- `a.media.chapter.name`

**Heartbeat:**
- `s:stream:chapter_id`

**Context Data:**
- `a.media.chapter.name`

**Data Feed:** videochapter

**Audience Manager:** `c_contextdata.a.media.chapter.name`

The auto-generated ID of the chapter.

### Chapter Metrics

#### l:stream:chapter_length

- **Required:**
- **SDK:** No
- **API:** Yes
- **Type:** number
- **Sent with:** Chapter Close
- **Min. SDK Version:** 1.3
- **Sample value:**

**Adobe Analytics:**
- `a.media.chapter.view`

**Heartbeat:**
- `s:event:type=chapter_start`

**Context Data:**
- `a.media.chapter.view`

**Data Feed:** videochapterstart

**Audience Manager:** `c_contextdata.a.media.chapter.view`

The number of chapter starts.

**Important:** If this event is set, the only possible value is TRUE. If this event is not set, no value is sent.
### Quality parameters

This topic presents a list of quality of experience (QoE/QoS) data, including context data values, that Adobe collects via solution variables.

This section contains the following information:

- **Quality Metadata**
- **Quality Metrics**

---

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
</table>
| **Chapter Complete** | • SDK Key: *Automatically set*  
• API Key: N/A  
• Required: Yes  
• Type: string  
• Sent with: Chapter Close  
• Min. SDK Version: 1.3  
• Sample value: TRUE | • *Adobe Analytics*: a.media.chapter.complete  
• *Heartbeat*: s:event:type=chapter_complete | • Available: Yes  
• Reserved Variable: event  
• Report Name: *Chapter Completes*  
• Context Data: a.media.chapter.complete  
• Data Feed: videochaptercomplete  
• Audience Manager: c_contextdata.a.media.chapter.complete |

The number of chapter completes.

*Important:* If this event is set, the only possible value is TRUE. If this event is not set, no value is sent.

| Chapter Time Spent | **Release Date: 09/13/18**  
The time spent on the chapter.  
The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports & Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds. | **Available**: Yes  
**Reserved Variable**: event  
**Report Name**: *Chapter Time Spent*  
**Context Data**: a.media.chapter.timePlayed  
**Data Feed**: videochaptertime  
**Audience Manager**: c_contextdata.a.media.chapter.timePlayed |

---

**Quality parameters**

This topic presents a list of quality of experience (QoE/QoS) data, including context data values, that Adobe collects via solution variables.

This section contains the following information:

- **Quality Metadata**
- **Quality Metrics**
Table data description:

- **Label**: The name of the parameter.
- **Implementation**: Information on implementation values and requirements
  - **Key**: Variable, set either manually in your app, or automatically by the Adobe Media SDK.
  - **Required**: Indicates whether the parameter is required for basic video tracking.
  - **Type**: Specifies the type of the variable to be set, string or number.
  - **Sent With**: Indicates when the data is sent: *Initiate* is the analytics call sent on video start, *Ad Start* is the analytics call sent on ad start, *Chapter Start* is the analytics call sent on chapter start, and *Close* is the compiled analytics call sent directly from the heartbeat server to the analytics server at the end of the media session, or the end of the ad. The Close calls are not available in network packet calls.
  - **Min. SDK Version**: Indicates which SDK version you would need to access the parameter.
  - **Sample Value**: Provides example of common variable usage.

- **Network Parameters**: Displays the values that are passed to Adobe Analytics or Heartbeat servers. This column shows the names of the parameters that are seen in the network calls generated by Adobe Media SDKs.
- **Reporting**: Information on how to view and analyze the video data.
  - **Available**: Indicates whether the data is available in reporting by default (*Yes*), or requires custom set-up (*Custom*).
  - **Reserved Variable**: Indicates whether the data is captured as an event, eVar, prop, or classification in a reserved variable.
  - **Report Name**: Name of Adobe Analytics report for variable
  - **Context Data**: Name of the Adobe Analytics context data passed to the reporting server and used in processing rules.
  - **Data Feed**: Column name for variable found in Clickstream or Live Stream data feeds
  - **Audience Manager**: Trait name found in Adobe Audience Manager

### Quality Metadata

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
</table>
| **Average Bitrate** | - **SDK Key**: bitrate*  
|                | - **API Key**: media.qoe.bitrate  
|                | - **Required**: No  
|                | - **Type**: number  
|                | - **Sent with**: Media Close  
|                | - **Min. SDK Version**: Any  
|                | - **Sample value**: 800–899  
| **Adobe Analytics**:  
|                | a.media.qoe.bitrateAverageBucket  
| **Heartbeat**:  
|                | l:stream:bitrate  
|                | **Available**: Yes  
|                | **Reserved Variable**: eVar  
|                | **Expiration**: On HIT  
|                | **Report Name**: Average Bitrate  
|                | **Context Data**:  
|                | a.media.qoe.bitrateAverageBucket  
|                | **Data Feed**:  
|                | videoqoebitrateaverageevar  
|                | **Audience Manager**:  
|                | c_contextdata.a.media.qoe.bitrateAverageBucket  

The average bitrate (in kbps). The value is predefined buckets at 100kbps intervals. The Average Bitrate is computed as a weighted average of all bitrate values related to the play duration that occurred during a playback session.
### Time to Start

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createQoSObject</code></td>
<td><em>(bitrate, startupTime, fps, droppedFrames)</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **SDK Key:** `startupTime`<br>- **API Key:** `media.qoe.timeToStart`<br>- **Required:** No<br>- **Type:** number<br>- **Sent with:** Media Start, Media Close<br>- **Min. SDK Version:** Any<br>- **Sample value:** 30,000 (milliseconds)<br>- **Available:** Yes<br>- **Reserved Variable:** eVar<br>- **Expiration:** On HIT<br>- **Report Name:** Time to Start<br>- **Context Data:** `a.media.qoe.timeToStart`<br>- **Data Feed:** `videoqoetimetostartevar`<br>- **Audience Manager:** `c_contextdata.a.media.qoe.timeToStart`<br><br>This value defaults to zero if you do not set it through the QoSObject. You set this value in milliseconds. The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports & Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds.<br>* `createQoSObject` *(bitrate, `startupTime`, fps, droppedFrames)*

### FPS

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createQoSObject</code></td>
<td><em>(bitrate, startupTime, fps, droppedFrames)</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **SDK Key:** `fps`<br>- **API Key:** `media.qoe.framesPerSecond`<br>- **Required:** No<br>- **Type:** number<br>- **Sent with:** Media Start, Media Close<br>- **Min. SDK Version:** Any<br>- **Sample value:** 24<br>- **Available:** No<br>- **Reserved Variable:** N/A<br>- **Report Name:** N/A<br>- **Context Data:**<br>- **Data Feed:**<br>- **Audience Manager:**<br><br>The current value of the stream frame-rate (in frames per second).<br>* `createQoSObject` *(bitrate, startupTime, `fps`, droppedFrames)*

### Dropped Frames

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createQoSObject</code></td>
<td><em>(bitrate, startupTime, fps, droppedFrames)</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **SDK Key:** `droppedFrames`<br>- **API Key:** `media.qoe.droppedFrames`<br>- **Required:** No<br>- **Type:** number<br>- **Sent with:** Media Close<br>- **Min. SDK Version:** Any<br>- **Sample value:** 3<br>- **Available:** Yes<br>- **Reserved Variable:** eVar<br>- **Expiration:** On HIT<br>- **Report Name:** Dropped Frames<br>- **Context Data:** `a.media.qoe.droppedFrameCount`<br>- **Data Feed:** `videoqoedroppedframecountevar`
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>Audience Manager:</strong></td>
<td></td>
<td></td>
<td>c_contextdata.a.media.qoe.droppedFrameCount</td>
</tr>
<tr>
<td></td>
<td>The number of dropped frames (Integer). This value is computed as a sum of all frames dropped during a playback session.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This value is taken from the last value of l:stream:dropped_frames.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>createQoSObject</em>(bitrate, startupTime, fps, droppedFrames)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Buffer Events</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• <strong>SDK Key:</strong> Automatically set</td>
<td>• <strong>Adobe Analytics:</strong></td>
<td>• <strong>Available:</strong> Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>API Key:</strong> N/A</td>
<td>• <strong>Reserved Variable:</strong> eVar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Required:</strong> No</td>
<td>• <strong>Expiration:</strong> On HIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Type:</strong> number</td>
<td>• <strong>Report Name:</strong> Buffer Events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Sent with:</strong> Media Close</td>
<td>• <strong>Context Data:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Min. SDK Version:</strong> Any</td>
<td>a.media.qoe.bufferCount</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Sample value:</strong> 2</td>
<td>s:event:type=buffer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The number of buffer events. This metric is computed as a count of the different buffer states that occurred during a playback session.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This is a count of how many times the player enters a buffer state from other states, e.g., playing or pausing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Buffer Duration</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• <strong>SDK Key:</strong> Automatically set</td>
<td>• <strong>Adobe Analytics:</strong></td>
<td>• <strong>Available:</strong> Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>API Key:</strong> N/A</td>
<td>• <strong>Reserved Variable:</strong> eVar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Required:</strong> No</td>
<td>• <strong>Expiration:</strong> On HIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Type:</strong> number</td>
<td>• <strong>Report Name:</strong> Total Buffer Duration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Sent with:</strong> Media Close</td>
<td>• <strong>Context Data:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Min. SDK Version:</strong> 30 (Seconds)</td>
<td>a.media.qoe.bufferTime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Sample value:</strong> 30 (Seconds)</td>
<td>s:event:duration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The total amount of time, in seconds, spent buffering. This value is computed as a sum of all buffer events durations that occurred during a playback session.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Release Date: 09/13/18**
### Reporting

The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports & Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds.

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
</table>
| **Bitrate Changes** | • Adobe Analytics:  
  `a.media.qoe.bitrateChangeCount`  
  **Reserved Variable**: eVar  
  **Expiration**: On HIT  
  **Report Name**: *Bitrate Changes*  
  **Context Data**:  
  `a.media.qoe.bitrateChangeCount`  
  **Data Feed**:  
  `videoqoe.bitratechangecountevar`  
  **Audience Manager**:  
  `c_contextdata.a.media.qoe.bitrateChangeCount` | | • Available: Yes  
• Reserved Variable: eVar  
• Expiration: On HIT  
• Report Name: *Bitrate Changes*  
• Context Data:  
  `a.media.qoe.bitrateChangeCount`  
• Data Feed:  
  `videoqoe.bitratechangecountevar`  
• Audience Manager:  
  `c_contextdata.a.media.qoe.bitrateChangeCount` |
| • SDK Key:  
  `media.qoe.bitrateChange`  
• **Required**: No  
• **Type**: number  
• Sent with: Media Close  
• Min. SDK Version: Any  
• Sample value: 3 | | | |

The number of bitrate changes (Integer). This value is computed as a sum of all bitrate change events that occurred during a playback session.

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
</table>
| **Errors / Error Events** | • Adobe Analytics:  
  `a.media.qoe.errorCount`  
  **Reserved Variable**: eVar  
  **Expiration**: On HIT  
  **Report Name**: *Errors*  
  **Context Data**:  
  `a.media.qoe.errorCount`  
  **Data Feed**:  
  `videoqoe.errorcountevar`  
  **Audience Manager**:  
  `c_contextdata.a.media.qoe.errorCount` | | • Available: Yes  
• Reserved Variable: eVar  
• Expiration: On HIT  
• Report Name: *Errors*  
• Context Data:  
  `a.media.qoe.errorCount`  
• Data Feed:  
  `videoqoe.errorcountevar`  
• Audience Manager:  
  `c_contextdata.a.media.qoe.errorCount` |
| • SDK Key:  
  `a.media.qoe.errorCount`  
• **Required**: No  
• **Type**: number  
• Sent with: Media Close  
• Min. SDK Version: Any  
• Sample value: 1 | | | |

The number of errors occurred (Integer). This value is computed as a sum of all error events that occurred during a playback session.

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
</table>
| **Player SDK Error IDs** | • Adobe Analytics:  
  `a.media.qoe.playerSdkErrors`  
  **Reserved Variable**: eVar  
  **Expiration**: On HIT  
  **Report Name**: *Errors*  
  **Context Data**:  
  `a.media.qoe.playerSdkErrors`  
  **Data Feed**:  
  `videoqoe.playersdkerrors`  
  **Audience Manager**:  
  `c_contextdata.a.media.qoe.playerSdkErrors` | | • Available: Yes  
• Reserved Variable: eVar  
• Expiration: On HIT  
• Report Name: *Errors*  
• Context Data:  
  `a.media.qoe.playerSdkErrors`  
• Data Feed:  
  `videoqoe.playersdkerrors`  
• Audience Manager:  
  `c_contextdata.a.media.qoe.playerSdkErrors` |
| • SDK Key: Automatically set  
• **Required**: No  
• **Type**: number  
• Sent with: Media Close  
• Min. SDK Version: Any  
• Sample value: | | | |
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Error IDs</td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: a.media.qoe.externalErrors</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key:</td>
<td>• Heartbeat: s:event:type=error</td>
<td>• Reserved Variable: eVar</td>
</tr>
<tr>
<td></td>
<td>• Required: No</td>
<td>• Expiration: On HIT</td>
<td>• Report Name: Errors</td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td>• Context Data: a.media.qoe.externalErrors</td>
<td>• Context Data:</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td>• Data Feed: videoqoeexternalerrors</td>
<td>• Data Feed:</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td>• Audience Manager: c_contextdata.a.media.qoe.externalErrors</td>
<td>• Audience Manager:</td>
</tr>
<tr>
<td></td>
<td>• Sample value:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Media SDK Error IDs</td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: a.media.qoe.mediaSdkErrors</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key:</td>
<td>• Heartbeat: s:event:type=error</td>
<td>• Reserved Variable: eVar</td>
</tr>
<tr>
<td></td>
<td>• Required: No</td>
<td>• Expiration: On HIT</td>
<td>• Report Name: Errors</td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td>• Context Data: a.media.qoe.mediaSdkErrors</td>
<td>• Context Data:</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td>• Data Feed: mediaqoeexntneralerrors</td>
<td>• Data Feed:</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td>• Audience Manager:</td>
</tr>
<tr>
<td></td>
<td>• Sample value:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session End</td>
<td>• SDK Key: Automatically set</td>
<td>• Adobe Analytics: N/A</td>
<td>• Available: Use custom</td>
</tr>
<tr>
<td></td>
<td>• API Key:</td>
<td>• Heartbeats: s:event:type=end</td>
<td>processing rule</td>
</tr>
<tr>
<td></td>
<td>• Type: string</td>
<td>• Report Name: N/A</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td>• Context Data:</td>
<td>• Report Name: N/A</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: 2.1</td>
<td>• Data Feed:</td>
<td>• Context Data:</td>
</tr>
<tr>
<td></td>
<td>• Sample value: end</td>
<td></td>
<td>• Data Feed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Audience Manager:</td>
</tr>
</tbody>
</table>

The unique error IDs generated by the player SDK. Customers must provide the error codes/ids at implementation time via provided error APIs.

The unique error IDs from any external source, e.g., CDN errors. Customers must provide the error codes/ids at implementation time via provided error APIs.

The unique error IDs generated by Media SDK during playback.

The end event means that the SDK is sending a close call to the backend. On the receipt of this event, the backend will close the session for this video, and do no further processing.
## Quality Metrics

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time To Start</strong></td>
<td>• SDK Key: <em>Automatically set</em></td>
<td>• Adobe Analytics:</td>
<td>• Available: Yes</td>
</tr>
<tr>
<td></td>
<td>• API Key: N/A</td>
<td>a.media.qoe.timeToStart</td>
<td>• Reserved Variable: event</td>
</tr>
<tr>
<td></td>
<td>• Required: No</td>
<td>• Heartbeat:</td>
<td>• Report Name: <em>Time to Start</em></td>
</tr>
<tr>
<td></td>
<td>• Type: number</td>
<td>l:stream:startup_time</td>
<td>• Context Data:</td>
</tr>
<tr>
<td></td>
<td>• Sent with: Media Close</td>
<td></td>
<td>a.media.qoe.timeToStart</td>
</tr>
<tr>
<td></td>
<td>• Min. SDK Version: Any</td>
<td></td>
<td>• Data Feed:</td>
</tr>
<tr>
<td></td>
<td>• Sample value: 30,000 (milliseconds)</td>
<td></td>
<td>videoqoetimetostart</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Audience Manager:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c_contextdata.a.media.qoe.timeToStart</td>
</tr>
</tbody>
</table>

*Release Date: 09/13/18*

This value defaults to zero if you do not set it through the QoSObject. You set this value in milliseconds. The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports & Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds.

*createQoSObject(bitrate, *startupTime*, fps, droppedFrames)*

| **Buffer Events** | • SDK Key: *startupTime*                            | • Adobe Analytics:                                       | • Available: Yes                               |
|                  | • API Key: N/A                                       | a.media.qoe.bufferCount                                 | • Reserved Variable: event                    |
|                  | • Required: No                                       | • Heartbeat:                                            | • Report Name: *Buffer Events*                 |
|                  | • Type: number                                       | s:event:type=buffer                                    | • Context Data:                                |
|                  | • Sent with: Media Close                             |                                                         | a.media.qoe.bufferCount                       |
|                  | • Min. SDK Version: Any                             |                                                         | • Data Feed:                                  |
|                  | • Sample value: 2                                   |                                                         | videoqoebuffercount                           |
|                  |                                                      |                                                         | • Audience Manager:                           |
|                  |                                                      |                                                         | c_contextdata.a.media.qoe.bufferCount         |

The number of buffer events (Integer). This metric is computed as a count of buffer events that occurred during a playback session.

<p>| <strong>Total Buffer Duration</strong> | • SDK Key: <em>Automatically set</em>                        | • Adobe Analytics:                                       | • Available: Yes                               |
|                          | • API Key: N/A                                       | a.media.qoe.bufferTime                                  | • Reserved Variable: event                    |
|                          | • Required: No                                       | • Heartbeat:                                            | • Report Name: <em>Total Buffer Duration</em>         |
|                          | • Type: number                                       | l:event:duration                                        | • Context Data:                                |
|                          | • Sent with: Media Close                             |                                                         | a.media.qoe.bufferTime                        |
|                          | • Min. SDK Version: Any                             |                                                         | • Data Feed:                                  |
|                          | • Sample value: 15                                  |                                                         | videoqoebuffertime                            |
|                          |                                                      |                                                         | • Audience Manager:                           |
|                          |                                                      |                                                         | c_contextdata.a.media.qoe.bufferTime          |</p>
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Release Date:</strong> 09/13/18</td>
<td>The total amount of time spent buffering (seconds; integer). This value is computed as a sum of all buffer events durations that occurred during a playback session. The value will be displayed in the time format (HH:MM:SS) in Analysis Workspace and Reports &amp; Analytics. In Data Feeds, Data Warehouse, and Reporting APIs the values will be displayed in seconds.</td>
<td><img src="https://via.placeholder.com/150" alt="Image" /></td>
<td><img src="https://via.placeholder.com/150" alt="Image" /></td>
</tr>
</tbody>
</table>
| **Bitrate Changes** | • Available: Yes  
  • Adobe Analytics: a.media.qoe.bitrateChangeCount  
  • Heartbeat: s:event:type=bitrate_change  
  • Reserved Variable: event  
  • Report Name: Bitrate Changes  
  • Context Data: a.media.qoe.bitrateChangeCount  
  • Data Feed: videoqebitratechangecount  
  • Audience Manager: c_contextdata.a.media.qoe.bitrateChangeCount | ![Image](https://via.placeholder.com/150) | ![Image](https://via.placeholder.com/150) |
| **Errors** | • Available: Yes  
  • Adobe Analytics: a.media.qoe.errorCount  
  • Heartbeat: s:event:type=error  
  • Reserved Variable: event  
  • Report Name: Error Events  
  • Context Data: a.media.qoe.errorCount  
  • Data Feed: videoqoeerrorcount  
  • Audience Manager: c_contextdata.a.media.qoe.errorCount | ![Image](https://via.placeholder.com/150) | ![Image](https://via.placeholder.com/150) |
| **Dropped Frames** | • Available: Yes  
  • Adobe Analytics: a.media.qoe.droppedFrameCount  
  • Heartbeat: l:stream:dropped_frames  
  • Reserved Variable: event  
  • Report Name: Dropped Frames  
  • Context Data: a.media.qoe.droppedFrameCount  
  • Data Feed: videoqoedroppedframecount  
  • Audience Manager: c_contextdata.a.media.qoe.droppedFrameCount | ![Image](https://via.placeholder.com/150) | ![Image](https://via.placeholder.com/150) |
### Reporting

**Network Parameters**

The number of dropped frames (Integer). This value is computed as a sum of all frames dropped during a playback session.

<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Drops Before Start</strong></td>
<td>• Adobe Analytics: a.media.qoe.dropBeforeStart&lt;br&gt;• Heartbeat: s:event:type=aa_start</td>
<td>• Available: Yes&lt;br&gt;• Reserved Variable: event&lt;br&gt;• Report Name: Drops before Start&lt;br&gt;• Context Data: a.media.qoe.dropBeforeStart&lt;br&gt;• Data Feed: videoqoedropbeforestart&lt;br&gt;• Audience Manager: c_contextdata.a.media.qoe.dropBeforeStart</td>
</tr>
<tr>
<td></td>
<td>• SDK Key: Automatically set&lt;br&gt;• API Key: N/A&lt;br&gt;• Required: No&lt;br&gt;• Type: string&lt;br&gt;• Sent with: Media Close&lt;br&gt;• Min. SDK Version: Any&lt;br&gt;• Sample value: TRUE</td>
<td></td>
</tr>
</tbody>
</table>

The number of times a user quit the video before its start. This metric is set to 1 only if no content was rendered, regardless of ads.

**Important:** If this event is set, the only possible value is TRUE. If this event is not set, no value is sent.

<table>
<thead>
<tr>
<th><strong>Buffer Impacted Streams</strong></th>
<th>• Adobe Analytics: a.media.qoe.buffer&lt;br&gt;• Heartbeat: s:event:type=buffer</th>
<th>• Available: Yes&lt;br&gt;• Reserved Variable: event&lt;br&gt;• Report Name: Buffer Impacted Streams&lt;br&gt;• Context Data: a.media.qoe.buffer&lt;br&gt;• Data Feed: videoqoebuffer&lt;br&gt;• Audience Manager: c_contextdata.a.media.qoe.buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• SDK Key: Automatically set&lt;br&gt;• API Key: N/A&lt;br&gt;• Required: No&lt;br&gt;• Type: string&lt;br&gt;• Sent with: Media Close&lt;br&gt;• Min. SDK Version: Any&lt;br&gt;• Sample value: TRUE</td>
<td></td>
</tr>
</tbody>
</table>

The number of streams impacted by buffering. This metric is set to 1 only if at least one buffer event occurred during a playback session.

**Important:** If this event is set, the only possible value is TRUE. If this event is not set, no value is sent.

<table>
<thead>
<tr>
<th><strong>Bitrate Change Impacted Streams</strong></th>
<th>• Adobe Analytics: a.media.qoe.bitrateChange&lt;br&gt;• Heartbeat: s:event:type=bitrate_change</th>
<th>• Available: Yes&lt;br&gt;• Reserved Variable: event&lt;br&gt;• Report Name: Buffer Change Impacted Streams</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• SDK Key: Automatically set&lt;br&gt;• API Key: N/A&lt;br&gt;• Required: No&lt;br&gt;• Type: string</td>
<td></td>
</tr>
</tbody>
</table>
### Reporting Network Parameters Implementation

- **Label**: Sent with: Media Close
- **Min. SDK Version**: Any
- **Sample value**: TRUE

**Network Parameters**

- **Context Data**: a.media.qoe.bitrateChange
- **Data Feed**: videoqoe bitratechange
- **Audience Manager**: c_contextdata.a.media.qoe.bitrateChange

---

The number of streams in which bitrate changes occurred. This metric is set to 1 only if at least one bitrate change event occurred during a playback session.

**Important**: If this event is set, the only possible value is TRUE. If this event is not set, no value is sent.

### Average Bitrate

- **SDK Key**: Automatically set
- **API Key**: N/A
- **Required**: No
- **Type**: number
- **Sent with**: Media Close
- **Min. SDK Version**: Any
- **Sample value**: 3200

**Adobe Analytics**

- **Available**: Yes
- **Reserved Variable**: event
- **Report Name**: Average Bitrate
- **Context Data**: a.media.qoe.bitrateAverage
- **Data Feed**: videoqoe bitrateaverage
- **Audience Manager**: c_contextdata.a.media.qoe.bitrateAverage

The average bitrate (in kbps, integer). This metric is computed as a weighted average of all bitrate values related to the play duration that occurred during a playback session.

### Error Impacted Streams

- **SDK Key**: Automatically set
- **API Key**: N/A
- **Required**: No
- **Type**: string
- **Sent with**: Media Close
- **Min. SDK Version**: Any
- **Sample value**: TRUE

**Adobe Analytics**

- **Available**: Yes
- **Reserved Variable**: event
- **Report Name**: Error Impacted Streams
- **Context Data**: a.media.qoe.error
- **Data Feed**: videoqoe error
- **Audience Manager**: c_contextdata.a.media.qoe.error

The number of streams in which bitrate changes occurred. This metric is set to 1 only if at least one bitrate change event occurred during a playback session.

**Important**: If this event is set, the only possible value is TRUE. If this event is not set, no value is sent.
<table>
<thead>
<tr>
<th>Label</th>
<th>Implementation</th>
<th>Network Parameters</th>
<th>Reporting</th>
</tr>
</thead>
</table>
| **Dropped Frame Impacted Streams** | • SDK Key: *Automatically set*  
• API Key: N/A  
• Required: No  
• Type: string  
• Sent with: Media Close  
• Min. SDK Version: Any  
• Sample value: TRUE | • Adobe Analytics:  
a.media.qoe.droppedFrames  
• Heartbeat:  
l:stream:dropped_frames | • Available: Yes  
• Reserved Variable: event  
• Report Name: *Dropped Frame Impacted Streams*  
• Context Data:  
a.media.qoe.droppedFrames  
• Data Feed:  
videoqoedroppedframes  
• Audience Manager:  
c_contextdata.a.media.qoe.droppedFrames |

The number of streams in which frames were dropped. This metric is set to 1 only if at least one frame was dropped during a playback session.

⚠️ **Important:** If this event is set, the only possible value is TRUE. If this event is not set, no value is sent.

| **Stalling Impacted Streams** | • SDK Key: *Automatically set*  
• API Key: N/A  
• Required: No  
• Type: string  
• Sent with: Media Close  
• Min. SDK Version: 1.5+  
• Sample value: TRUE | • Adobe Analytics:  
a.media.qoe.stall  
• Heartbeat:  
s:event:type=stall | • Available: Use custom processing rule  
• Reserved Variable: event  
• Report Name:  
• Data Feed: N/A  
• Context Data:  
a.media.qoe.stall  
• Audience Manager:  
c_contextdata.a.media.qoe.stall |

The number of streams in which a stalled event occurred. This metric is set to 1 only if at least one stall occurred during playback. Customers will have to create their own processing rules to have the value available for reporting.

⚠️ **Important:** If this event is set, the only possible value is TRUE. If this event is not set, no value is sent.

| **Stalling Events** | • SDK Key: *Automatically set*  
• API Key: N/A  
• Required: No  
• Type: string  
• Sent with: Media Close  
• Min. SDK Version: 1.5+ | • Adobe Analytics:  
a.media.qoe.stallCount  
• Heartbeat:  
s:event:type=stall | • Available: Use custom processing rule  
• Reserved Variable: event  
• Report Name:  
• Context Data:  
a.media.qoe.stallCount  
• Data Feed: N/A |
The number of times the playback was stalled during a playback session. Customers will have to create their own processing rules to have the value available for reporting.

<table>
<thead>
<tr>
<th>Total Stalling Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SDK Key:</strong> Automatically set</td>
</tr>
<tr>
<td><strong>API Key:</strong> N/A</td>
</tr>
<tr>
<td><strong>Required:</strong> No</td>
</tr>
<tr>
<td><strong>Type:</strong> number</td>
</tr>
<tr>
<td><strong>Sent with:</strong> Media Close</td>
</tr>
<tr>
<td><strong>Min. SDK Version:</strong> 1.5+</td>
</tr>
<tr>
<td><strong>Sample value:</strong> 12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adobe Analytics:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.media.qoe.stallTime</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heartbeat:</th>
</tr>
</thead>
<tbody>
<tr>
<td>s:event:type=stall</td>
</tr>
</tbody>
</table>

| Available: Use custom processing rule |
| Reserved Variable: event |
| Report Name: |
| Context Data: a.media.qoe.stallTime |
| Data Feed: N/A |
| Audience Manager: c_contextdata.a.media.qoe.stallTime |

The total time (seconds; integer) the playback was stalled during a playback session. Customers will have to create their own processing rules to have the value available for reporting.

### Segments

*Note: These reporting segments associated with Media Stream Type were introduced on 9/13/18 along with the streamType parameter.*

<table>
<thead>
<tr>
<th>Segment</th>
<th>Description</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Stream Type: All</td>
<td>Segment all media stream data</td>
<td>&quot;Content (ID) exists&quot;</td>
</tr>
<tr>
<td>Media Stream Type: Audio</td>
<td>Segment all audio stream data</td>
<td>&quot;Content (ID) exists&quot; AND &quot;Media Stream Type = audio&quot;</td>
</tr>
<tr>
<td>Media Stream Type: Video</td>
<td>Segment all video stream data</td>
<td>&quot;Content (ID) exists&quot; AND &quot;Media Stream Type != audio&quot;</td>
</tr>
<tr>
<td>Media Content Type: VoD</td>
<td>Segment all VoD contents</td>
<td>&quot;Content Type = vod&quot;</td>
</tr>
<tr>
<td>Media Content Type: Live</td>
<td>Segment all Live contents</td>
<td>&quot;Content Type = live&quot;</td>
</tr>
<tr>
<td>Media Content Type: Linear</td>
<td>Segment all Linear contents</td>
<td>&quot;Content Type = linear&quot;</td>
</tr>
<tr>
<td>Media Content Type: Podcast</td>
<td>Segment all Podcast contents</td>
<td>&quot;Content Type = podcast&quot;</td>
</tr>
<tr>
<td>Media Content Type: Audiobook</td>
<td>Segment all Audiobook contents</td>
<td>&quot;Content Type = audiobook&quot;</td>
</tr>
<tr>
<td>Media Content Type: AoD</td>
<td>Segment all AoD contents</td>
<td>&quot;Content Type = aod&quot;</td>
</tr>
</tbody>
</table>
## Calculated metrics

*Note: These calculated metrics were introduced on 9/13/18.*

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Ads per Media Stream</td>
<td>Ad Starts per Media Starts</td>
<td>Ad Starts / Media Starts</td>
</tr>
<tr>
<td>Avg. Chapters per Media Stream</td>
<td>Chapter Starts per Media Starts</td>
<td>Chapter Start / Media Starts</td>
</tr>
<tr>
<td>Avg. Media Time Spent</td>
<td>Total Time Spent per Media Starts (HH:MM:SS)</td>
<td>Media Time Spent / Media Starts</td>
</tr>
<tr>
<td>Avg. Content Time Spent</td>
<td>Content Time Spent per Content Starts (HH:MM:SS)</td>
<td>Content Time Spent / Content Start</td>
</tr>
<tr>
<td>Avg. Ad Time Spent</td>
<td>Ad time spent per Ad Starts (HH:MM:SS)</td>
<td>Ad Time Spent / Ad Start</td>
</tr>
<tr>
<td>Avg. Chapter Time Spent</td>
<td>Chapter Time Spent per Chapter Starts (HH:MM:SS)</td>
<td>Chapter Time Spent / Chapter Start</td>
</tr>
<tr>
<td>Media Completion Rate</td>
<td>Rate of Content Completed vs Media Starts (%)</td>
<td>Content Completes / Media Starts</td>
</tr>
<tr>
<td>Content Completion Rate</td>
<td>Rate of Content Completed vs Content Starts (%)</td>
<td>Content Completes / Content Starts</td>
</tr>
<tr>
<td>Ad Completion Rate</td>
<td>Rate of Ad Completes vs Ad Starts (%)</td>
<td>Ad Completes / Ad Starts</td>
</tr>
<tr>
<td>Chapter Completion Rate</td>
<td>Rate of Chapter Completes vs Chapter Starts (%)</td>
<td>Chapter Completes / Chapter Starts</td>
</tr>
<tr>
<td>Drop Before Start Rate</td>
<td>Rate of Drops before Starts vs Media Starts (%)</td>
<td>Drops before Starts / Media Starts</td>
</tr>
<tr>
<td>Content Pause Duration Rate</td>
<td>Rate of Total Pause Duration vs Content Time Spent (%)</td>
<td>Total Pause Duration / Content Time Spent</td>
</tr>
<tr>
<td>Content Buffer Duration Rate</td>
<td>Rate of Total Buffer Duration vs Content Time Spent (%)</td>
<td>Total Buffer Duration / Content Time Spent</td>
</tr>
<tr>
<td>Content Time to Start Rate</td>
<td>Rate of Time to Start vs Content Time Spent (%)</td>
<td>Time to Start / Content Time Spent</td>
</tr>
<tr>
<td>Ad Time Spent Rate</td>
<td>Rate of Ad Time Spent vs Content Time Spent (%)</td>
<td>Ad Time Spent / Content Time Spent</td>
</tr>
</tbody>
</table>
Reporting and analysis

Media reports enablement

Each report suite that collects video metrics must be configured before video data is sent.

**Tip:** To take advantage of new capabilities, existing Media Analytics customers should re-enable media tracking for their RSIDs.

2. Select the report suite(s) where you are collecting video data and click *Edit Settings > Media Management > Media Reporting*.

![Edit Settings](image)

3. On the *Media Reporting* page, enable **Media Core**, and optionally enable **Media Ads**, **Media Chapters**, and **Media Quality**.

Media measurement includes the following modules:

- **Media Core**: Core video measurement is used for media content. This will use Solution (or Custom) eVars to keep track of Content, Content Type, Content Player Name, and Content Channel. Solution (or Custom) events will be used for Media Starts, Content Starts, Content Completes, and Content Time Spent.

- **Media Ads**: Media ad measurement is used for the measurement of ads within the video content. This will use Solution eVvars to measure Ad, Ad Player Name, Ad Pod, and Ad in Pod Position. Solution events will be used for Ad Starts, Ad Completes, Ad Time Spent, and Video Time Spent.

- **Media Chapters**: Video chapters measurement is used for the measurement of chapters. A chapter is a sub-division of content within a single video. This will use a Solution eVar to store the Chapter ID. Solution events will be used for Chapter Starts, Chapter Completes, and Chapter Time Spent. Additional chapter metadata of Chapter Name and Chapter Position will be provided as classifications of Chapter ID.

- **Media Quality**: Video quality measurement is used for measuring the quality of the content playback. This will use Solution eVars to store Time to Start, Buffer Events, Total Buffer Duration, Bitrate Switches, Average Bitrate, Errors, and Dropped Frames. Solution events will be used for Time to Start, Drops before Start, Buffer Impacted Streams, Buffer Events, Total Buffer Duration, Bitrate Change Impacted Streams, Bitrate Changes, Avg Bitrate, Error Impacted Streams, Error Events, Dropped Frame Impacted Streams, and Dropped Frames.
- **Video & Video Ad Metadata**: Metadata can be attached to a video and/or an ad to further describe and categorize that video/ad. Standardized video and ad metadata will be collected via solution variables and classifications. Values to include: Show, Season, Episode, Asset ID, Genre, First Air Date, First Digital Date, Content Rating, Originator, Network, Show Type, Ad Loads, MVPD, Authorized, Day Part, Media Session ID, Advertiser, Campaign ID, and Creative ID.

- **Audio & Audio Ad Metadata**: Metadata can be attached to audio and/or an ad to further describe and categorize that audio/ad. Standardized audio and ad metadata will be collected via solution variables and classifications. Values to include: Artist, Album, Label, Author, Publisher, Station, Show, Season, Episode, Asset ID, Genre, First Air Date, First Digital Date, Content Rating, Originator, Show Type, Ad Loads, Day Part, Media Session ID, Advertiser, Campaign ID, and Creative ID.

Enabling each module reserves a set of variables and creates a new set of reports. With the exception of Quality, there will be no data in reports unless the corresponding implementation has been completed. Implementing the Core module also implements the Quality module if you enable it.

If you are not yet tracking ads, chapters, or playback quality, you can enable additional options at any time.

4. Click **Save**.

If this report suite is already configured to collect media data, after you click **Save**, an additional configuration page is displayed. If you see the **Media Core measurement** page, continue to the next step.

5. **(Conditional)** On the **Media Core measurement** page, select to continue using custom variables or to use solution variables.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continue using custom variables.</td>
<td>• <strong>Pros</strong>: Content trending continues to work after migration.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Cons</strong>: Requires you to keep two custom eVars and three custom events allocated to video. You regain use of one custom eVar and one custom event.</td>
</tr>
<tr>
<td></td>
<td>To continue using custom variables:</td>
</tr>
<tr>
<td></td>
<td>1. Select <strong>Use Custom Variables</strong>, then click <strong>Save</strong>.</td>
</tr>
<tr>
<td></td>
<td>2. When prompted, map your current custom eVars and events and then click <strong>Save</strong>:</td>
</tr>
<tr>
<td>Migrate to solution variables.</td>
<td>• <strong>Pros</strong>: You regain use of three custom eVars and four custom events.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Cons</strong>: You lose all historical trending and comparison for media reports.</td>
</tr>
<tr>
<td></td>
<td>This means that you cannot trend content views or content time played for any dates before you migrated to heartbeats.</td>
</tr>
<tr>
<td></td>
<td><strong>Restriction</strong>: <em>Do not migrate to solution variables unless you are certain that you do not want to preserve this trending.</em></td>
</tr>
</tbody>
</table>

All customers should use solution variables and processing rules to put media data into existing props and eVars, only if they need to preserve historical continuity.

To migrate to solution variables:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Select <strong>Use Solution Variables</strong> and click <strong>Save</strong>.</td>
</tr>
</tbody>
</table>

⚠️ **Important:** Do not change the classification names for any variables listed in the **Metrics and metadata tables** that are described there under **Reporting/Reserved Variable** as “classification”.

The media classifications are defined when a report suite is enabled for media tracking. From time to time, Adobe adds new properties, and when this occurs, customers must re-enable their report suites to get access to the new media properties. During the update process Adobe determines whether the classifications are enabled by checking the names of the variables. If any of them is missing, Adobe adds the missing ones again.

# Media default reports

## Default reports overview

Adobe Analytics provides several reports and metrics to track media performance on your website.

In addition to the metrics and dimensions available when you enable each of the modules, there are three additional dashboard-style reports that become available when you enable the Media Core module. Enabling the Ads module also changes the appearance of some of these dashboard-style reports by adding additional metrics and filters.

Media reports are listed on the **Reports** tab. Navigate to > **Media** > **Media Engagement** and select from:

<table>
<thead>
<tr>
<th>Media Report</th>
<th>Description</th>
<th>Common Business Insights</th>
</tr>
</thead>
</table>
| **Media overview**  | Displays several aggregate measurements to quickly monitor that media is performing as expected. A graph displays media starts next to ad impressions to let you quickly view and compare these metrics. | • Totals for top metrics including content starts, completion rate, average time on stream, and average media items per visit.  
• Total content and ad starts for media filtered by device type or country. |
| **Media detail**    | Displays detailed metrics for all media items including starts, concurrent viewers, completion rate, play percentage, and ad impressions. | • Totals for top metrics including media starts, content and ad starts, and average content per visit.  
• Total content and ad starts for media filtered by device type or country. |
| **Media daypart**   | Displays content starts by time of day to let you quickly view when your audience is engaged. | • Audience engagement by time of day.  
• Audience engagement compared to previous date ranges. |
<p>| <strong>Media concurrent viewers</strong> | Displays concurrent viewers during one day. The data can be filtered by content, device type, or country. | • Per-minute audience engagement over a 24-hour interval. |</p>
<table>
<thead>
<tr>
<th>Media Report</th>
<th>Description</th>
<th>Common Business Insights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Events and Media Variables</td>
<td>Additional reports are available. Media metrics and dimensions are standard</td>
<td>• Media Conversion (Events that occur after media is viewed) by generating a report with</td>
</tr>
<tr>
<td></td>
<td>Analytics variables that can be reported directly and added to other Analytics</td>
<td>visits that include a content type of media.</td>
</tr>
<tr>
<td></td>
<td>reports.</td>
<td>• Next/previous media flow using the media name prop.</td>
</tr>
</tbody>
</table>

**Media overview**

The Media Overview dashboard is designed to let you monitor media across your site. The Media Overview display shows several aggregate measurements so you can quickly monitor that media is performing as expected. A graph displays content starts next to ad starts to let you quickly view these metrics for each media item.

![Media Overview Dashboard](image)

**Quick Filters**

Quickly display media metrics by device or geo country:

![Quick Filters](image)
Media Performance

Click-and-drag to zoom in, then hover to view granular metrics for specific media. Click to reset the view after you zoom.

Media detail

The Media Detail dashboard displays detailed metrics for all content, including concurrent viewers or listeners, content starts, completion rate, time spent, and ad starts.
Media daypart

The Media Daypart dashboard displays content starts by time of day to let you quickly view when your audience is engaged.

Media concurrent viewers

The Media Concurrent Viewers dashboard displays concurrent viewers during one day. The data can be filtered by content, device type, or country.

Tip: No data will be displayed if the selected interval is not an entire day.
Report features

Here are some features of this report:

- This is not in real time. It has normal Adobe Analytics latency.
- The report covers a 24 hour time frame. The x-axis is time-of-day based on the report suite time zone.
- This shows concurrent viewers at minute granularity.
- There is a *Media Concurrent Viewers Report* that shows how many viewers are watching or listening across all content.
- There is a Concurrent Viewers report within the *Media Detail* report that shows how many viewers are watching or listening to one specific media item.
- The report only works across one day.
- The customer can look at historical concurrent viewer reports (limited to a single day).

Limitations

Here are some limitations for this report:

- You cannot export the data, such as ReportBuilder.
- You cannot present the data in a table format.
- You cannot send a report via email.
- Even if you do not track ads, you have to re-enable media tracking and select the Media Ad module.
- This functionality will provide accurate data when using a heartbeats Library that has Pause tracking capabilities.

Get concurrent viewers JSON report data

You can extract concurrent viewers report data using the Web Services APIs:

- The Web Services API docs are available at: [https://www.adobe.io/apis/experiencecloud/analytics/docs.html](https://www.adobe.io/apis/experiencecloud/analytics/docs.html)

For example, you could filter the data using any segment that built on the UI. To filter by a specific content ID, you would need to create a new segment. A sample configuration will look like this:

```json
{
    "reportDescription": {
        "reportSuiteID": "nflmobileappprod",
        "dateFrom": "2018-07-08",
        "dateTo": "2018-07-09",
        "metrics": [
            {
                "id": "instances"
            }
        ],
        "elements": [
            {
                "id": "videoconcurrentviewers",
                "top": "2880"
            }
        ],
        "segments": [
            {
                "id": "sl234_58ca4fc7e4b0abc238707bb9"
            }
        ],
        "sortBy": "instances",
        "locale": "en_US"
    }
}
```
Media workspace templates

Adobe Analytics Workspace gives you the ability to quickly set up dashboards for analyzing your tracking data. When you create a new project, or when you go in to Workspace to manage an existing project, you can choose from standard templates that cover common general tracking scenarios, such as Acquisition, Audio and Video consumption, and more. You can also create, modify, and save your own custom templates.

You work with templates from the Workspace tab in Adobe Analytics, when you create or manage a project. When you initially go into Workspace, you are presented with the options of creating a new project or opening an existing project:

- **Create a new project** - When you click the Create New Project button in Workspace, you are presented with a dialog from which you can select from your collection of templates (including Standard and Custom):

  ![Workspace template selection](image)

  For example, if you choose the Audio Consumption template, a project with the following audio-oriented dashboard is presented:
• **Manage projects** - In an existing project, you can modify the data presentation and Save As to create your own collection of custom templates.
Federated Analytics

The Federated Analytics service provides a system for sharing Adobe Media (audio and video) Analytics data between two partners. The standardized measurement data created by Media Analytics is the hallmark for Federated Analytics, allowing the same data to flow into a single report from multiple sources. Through the rules and logic governing Federated Analytics, data is easily controlled and individualized to meet the needs of each partnership. Federated Analytics makes audio and video measurement more efficient, streamlined, and actionable.

Benefits

- **Transparent**: Strip away the black box of data creation by using the same logic across companies
- **Broad**: Understand the full reach and impact of audio and video consumption across partnerships, platforms, and devices
- **Secure**: Control server-side data sharing through rules and logic
- **Standardized**: Speak the same data language as your partners
- **Actionable**: Quantify audio and video data to benchmark players, monitor trends, and detect anomalies through Adobe Analytics
- **Centralized**: Collect audio and video measurement data in one Adobe location
- **Contractual**: Meet legal data sharing requirements easily
- **Timely**: Send and receive data in near real-time
- **Easy**: Tag players once with Adobe SDKs, share data to many partners

Definitions

- **Sender**: Customer generating audio and video analytics data on owned players
- **Receiver**: Customer receiving audio and video analytics data from sender

Requirements

- **Media Streams Contract**: Receiver and Sender must have contracted Adobe Analytics for Media Streams before gaining access to audio and video data within Adobe Analytics. Contact your account team for more details.
- **Federated Addendum**: Each Sender and Receiver must have a signed addendum in place with Adobe before sending or receiving data. One addendum per customer is required, not one addendum per partnership. Contact your account team for more details.
- **Media Analytics Implementation**: The Sender must have Media Analytics implemented on all players that will be part of the federated data set. Only Media Analytics data is available for federation. See documentation: [https://marketing.adobe.com/resources/help/en_US/sc/appmeasurement/hbvideo/](https://marketing.adobe.com/resources/help/en_US/sc/appmeasurement/hbvideo/)
- **Adobe Consulting Contract**: For initial set-up of federated rules between receiver and sender it is valuable to work with consulting services to review data and create the data sharing agreement.

Process

1. Sender and Receiver work together to complete the Federation Rules Agreement form.
Download the current version of the form here:

Note: This form contains special fields for our engineering team and should ONLY be edited using Adobe Acrobat. Download Acrobat for free.

2. Consulting services provides a sample data file to Receiver with actual data from Sender players, to further confirm correct data sharing rules are defined, provided data files are available.
3. Sender and Receiver ensure the data sharing agreement will meet all contractual requirements between the two parties.
4. Consulting services sends the completed form to Adobe Engineering to set-up data sharing rules.
5. Data is shared to the development report suite where Receiver will review and validate data.
6. Once Receiver confirms data is correct, Adobe Engineering updates the rules to point to a production report suite.
7. Receiver will review and validate data in the production report suite.
8. If changes occur to the data set in the future, Sender or Receiver can submit a customer care ticket for support.
Documentation updates

**Last updated:** February, 2019

- This update was primarily for the 2.2 Media SDK release on the JavaScript and OTT platforms.
- The 2.2 Media SDK release on JavaScript and OTT platforms provides the same support as described below for the iOS and Android platforms (November 2018 update).

**Update:** November 1, 2018

- This update was primarily for the 2.2 Media SDK release on the Android and iOS platforms.
- The 2.2 Media SDK release on Android and iOS provides support for tracking audio on those platforms, along with internal improvements.
- With the addition of audio tracking, and with both audio and video tracking capabilities now available in both the Media SDK and the Media Collection API, a relatively wholesale naming update is called for:
  - The overall solution is titled Adobe Analytics for Audio and Video
  - The shorthand formerly used throughout the docs, "Video Analytics", is now "Media Analytics"
  - In the SDK, references to "Video Heartbeat Library (VHL)" are now "Media SDK"
  - Filenames and URLs (e.g., links to API references) that formerly referenced "video" or "vhl" now use "media" in their place
  - In the code, the names of metadata keys now include "MEDIA" instead of "VIDEO"
  - and so on...

- Along with the above, some additional restructuring has occurred in the Media SDK section, including Standard Metadata implementation and reference returning to their own topics (they had been absorbed into the Track Core topics in the previous doc update). These topics, along with the Track core, and Track seeking, and Track buffering topics are now grouped together under Track audio and video playback.
- The Federated Analytics form was updated to Version 3.2, to reflect new parameters involved with tracking audio.

**Update:** October 10, 2018

- Document structure was "refactored" in the SDK Implementation area, by combining the individual (but mostly identical) platform implementation guides into one SDK implementation section, with platform-specific tracking examples presented in sub-sections beneath common tracking topics.
- Files were renamed throughout in anticipation of a migration to a new doc system. All DITA prefixes (c_, r_, t_ indicating concept, reference, and task topic types respectively) were eliminated. All underscores ("_") were replaced with hyphens ("-"). Also, file names now more closely resemble the titles of the topics.
- Updates to general Validation and Certification topics.
- New introductory material including a presentation of measurement options, along with updates to prerequisites, implementation paths, and Audience Manager enablement.
- Updates to Metrics and Metadata and Reporting and Analysis sections, reflecting the addition of Audio Analytics capabilities.
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 Experience 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 Experience 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

© 2018 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.