comscore

Android Library

Streaming Tag Implementation Guide

document version: 6.2.0; released on March 16, 2020

for further information, please contact:
Comscore
Tag Support
+1 866 276 6972

Contents

1 Introduction

Use of the Comscore SDK is subject to the licenses and other terms and conditions set forth herein, including the materials provided in the SDK deliverables. Your use of this SDK and/or transmission of data to Comscore constitutes your agreement to these licenses and other terms and conditions, including the Data Sharing Agreement.

The Android library Streaming Tag provides accurate and comprehensive streaming media analytics functionality. This enables comScore to receive measurement insights critical to answering questions about streaming media usage, including advertising messages.

The Android library Streaming Tag is implemented next to — or into — a media player in an Android application. In response to media change and playback state change activity in your player you will implement calls to the comScore library. A similar solution is available for other popular platforms from which Comscore reports streaming media usage.

If you have any questions or concerns about the instructions in this document, or about elements of the Android library, then please contact your Comscore account team or implementation support team.

1.1 Intended use of Android library

The instructions in this document are intended to be used with **version 6.3.0 and subsequent 6.x.y releases** of the Android library for implementation in a **media player in an Android application developed in Java** code. The release notes mention the Android OS versions which each Android library version can be used with. If your application is developed in another programming language then please contact your Comscore account team to ask for guidance.

1.2 Preparation

Please complete the following checklist before adding the Streaming Tag implementation to your media player in an Android application:

1. The Streaming Tag implementation uses elements of the Android library. Confirm you have implemented the library for tagging of the application itself.
2. Familiarize yourself with the instructions in this document.
3. If you are updating an existing implementation, then please refer to *Appendix D: Update an existing implementation on page 22* to see if there are any relevant steps mentioned for your situation.
4. Clarify with your comScore account team what type of media you should be implementing the Streaming Tag for (video and/or audio). Please do not implement this tag onto media types other than those you have been instructed to by your comScore account team.
5. Determine the media asset metadata values that need to be collected.
6. Make sure you are using a player that has an API which allows you to detect the player state and allows you to access details like the current playback position and relevant media asset metadata.

1.3 Implementation overview and general instructions

The implementation for a media player in an Android application involves the following steps:

1. Ensure the library is included in the application project with code statements to configure and start the library.
2. Create a `StreamingAnalytics` instance.
3. Specify media metadata values using `ContentMetadata` and `AdvertisementMetadata` instances.
4. Instrument the `StreamingAnalytics` instance so it is aware of media asset changes.
5. Instrument the `StreamingAnalytics` instance to make it aware of player playback state changes.

1.3.1 Intended use of library elements

As you work with the library you might see classes, methods or properties which do not appear in this documentation. Those library elements are exposed either because the solution requires it or because they are needed for custom solution implementations for which Comscore provides additional instructions.

Please ensure you do not use any library elements which do not appear in this documentation unless you have received explicit instructions for their use from Comscore.

1.3.2 Use `import` statements

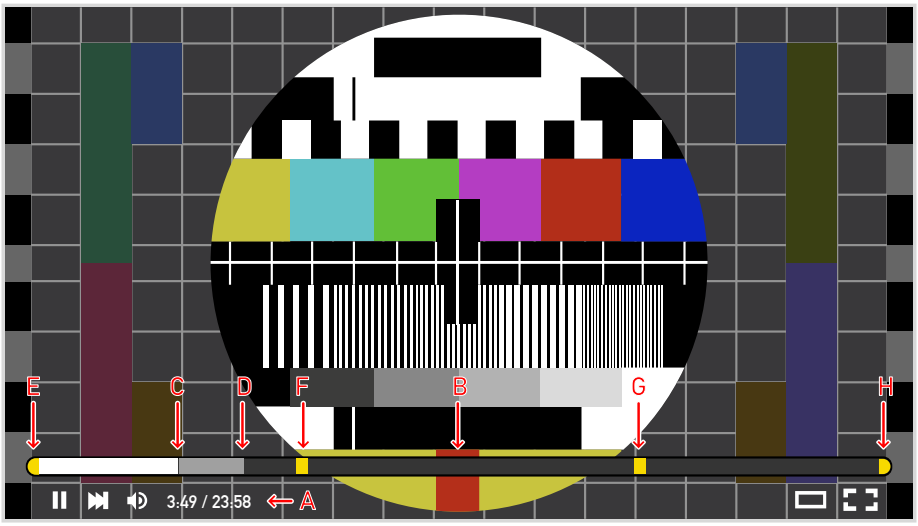
Your IDE (“Integrated Development Environment”) will likely take care of importing any classes you use from the `com.comscore.streaming` package. If it does not, or if you do not use an IDE then please make sure to include appropriate `import` statements to be able to use the class names without the need to specify the package.

2 Implementation instructions

For optimal tagging of your player's streaming playback scenarios it is important to understand how the Streaming Tag collects data. This data collection model can be summarized as follows:

- A *Playback Session* represents the collection of a discrete content and its related advertisements.
- Each discrete content is represented by exactly one *Asset*, specified through *Metadata* values.
- Each individual advertisement is represented by exactly one *Asset*, specified through *Metadata* values.
- Media changes in the player are indicated through an API method call to specify the metadata of the current *Asset*.
- The player's playback state changes — *play*, *pause*, *buffer*, etc. — are indicated through API method calls.

Please consider the following example player:



Example player with Time Line showing content and ad breaks

- **A** indicates the current *Playback Position* relative to the length of the content. The player is at position 3m49s of the content, which has a length of 23m58s.
- **B** is a visual representation of the *Time Line* for the content. It represents the content in its entirety and shows a number of relevant details which give an indication of what the player can be expected to do next.
- **C** visually represents the current *Playback Position* on the *Time Line*.
- **D** visually represents the amount of content data downloaded by the player. The player has not yet downloaded the entire content, so seeking to a position further into the content will likely cause buffering to occur. Or, seeking to a further position might not be possible altogether depending on how the player has been programmed to behave in such scenarios.
- **E**, **F**, **G** and **H** are cue point markers for ad breaks. At these positions - relative to the content - the player is potentially going to halt playback of the content to load and play advertisements.
  - **E** represents a pre-roll ad break.
  - **F** and **G** represent mid-roll ad breaks.
  - **H** represents a post-roll ad break.

If we assume the pre-roll, post-roll and first mid-roll ad break each contain a single individual advertisement and the second mid-roll ad break contains two individual advertisements, then this *Playback Session* has a total of 6 *Assets*:

- 1 content
- 1 pre-roll advertisement

- 3 mid-roll advertisements
- 1 post-roll advertisement

The following sections explain the implementation of the Streaming Tag in your player, illustrated with this example player.

2.1 Create `StreamingAnalytics` instance

To start, please create an instance of the `StreamingAnalytics` class from the comScore library:

```
11. StreamingAnalytics sa = new StreamingAnalytics();
```

You can reuse this instance throughout your implementation, even if your player changes from one content to another.

2.2 Set implementation details (optional)

To help with implementation validation and reporting Comscore may have provided you with additional instructions to identify your implementation and/or player.

2.2.1 Implementation ID

If Comscore provided you with an *Implementation ID* for your implementation, then please specify this ID as a `String` value:

```
12. sa.setImplementationId( "1234567890" ); // Use the provided ID
```

2.2.2 Project ID

If Comscore provided you with an *Project ID* for your implementation, then please specify this ID as a `String` value:

```
12. sa.setProjectId( "1234567890" ); // Use the provided ID
```

2.2.3 Player name and version

If Comscore instructed you to identify your players by name and version, then please specify these as `String` values:

```
13. sa.setMediaPlayerName( "My Player" ); // Use a suitable name to distinguish your player
14. sa.setMediaPlayerVersion( "1.2.3-a5f72c" ); // Use the version of your player
```

2.3 Create *Playback Session*

When your player loads content for playback — or the first time your player loads an advertisement related to that content — please instruct the `StreamingAnalytics` instance to create a new *Playback Session*:

```
21. sa.createPlaybackSession();
```

Advertisements that are played in relation to content should be in the same *Playback Session* as their related content. When advertisements are involved you would typically change the current *Playback Session* after any post-rolls and before any pre-rolls so that content and its related advertisements end up in the same *Playback Session*.

2.4 Specify *Asset* metadata

Each *Asset* is represented by metadata values. These metadata values are specified on `ContentMetadata.Builder` and `AdvertisementMetadata.Builder` object instances. These object instances then become `AdvertisementMetadata` and `ContentMetadata` objects by calling their `build` method as shown in the code examples in this section.

How to decide if the asset is content or advertisement...

In cases where defining a stream as advertisement or content is ambiguous, streams should be classified as content if they can be monetized. A stream can be monetized if it could (or did) have advertisements run against it. Conversely, a stream should be classified as advertisement if it is not in a position to have advertisements run against it due to the promotional nature of its subject matter.

The following types of video streams should **not** be tagged using the Streaming Tag unless otherwise directed by your Comscore account team.

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

2.4.1 Specify content metadata

Once the `ContentMetadata.Builder` instance is created, metadata values are specified using its API methods. The full list of available content metadata is provided in *Appendix A: Content metadata list on page 13*.

The following code example creates an instance of `ContentMetadata` and specifies those metadata values *required for Video Metrix tagging* to represent the content from our example:

```
31. ContentMetadata cm = new ContentMetadata.Builder()
32.     .mediaType( ContentType.LONG_FORM_ON_DEMAND )
33.     .uniqueId( "13784" )
34.     .length( 1418000 ) // 23m58s in milliseconds
35.     .dictionaryClassificationC3( "*null" )
36.     .dictionaryClassificationC4( "*null" )
37.     .dictionaryClassificationC6( "*null" )
38.     .stationTitle( "Hulu" )
39.     .publisherName( "ABC" )
40.     .programTitle( "Modern Family" )
41.     .genreName( "Comedy" )
```

```
    .classifyAsCompleteEpisode( true )
    .build()
    ;
```

2.4.2 Specify advertisement metadata

Once the `AdvertisementMetadata.Builder` instance is created, metadata values are specified using its API methods. The full list of available content metadata is provided in *Appendix B: Advertisement metadata list on page 19*.

The following code creates an instance of `AdvertisementMetadata.Builder` and specifies metadata values to represent the pre-roll advertisement from our example:

```
AdvertisementMetadata am = new AdvertisementMetadata.Builder()
    .mediaType( AdvertisementType.ON_DEMAND_PRE_ROLL )
    .relatedContentMetadata( cm )
    .build()
    ;
```

2.5 Add media change notifications

When your player loads content or advertisements for playback, you need to indicate which of the media metadata reflects what is currently loaded. For example, to indicate the player has currently loaded the pre-roll advertisement from our example:

```
sa.setMetadata( am );
```

Likewise, to indicate the player has currently loaded the content from our example:

```
sa.setMetadata( cm );
```

The `setMetadata` method accepts `AdvertisementMetadata` and `ContentMetadata` objects as its argument. To get these objects, the respective `AdvertisementMetadata.Builder` and `ContentMetadata.Builder` objects need to be built, by calling their `build` method as shown in the code examples in the previous section.

2.6 Add playback state change notifications

As your player plays content and advertisements, it will go through one or more of the playback state changes listed below. Please implement calls the associated notification methods on the `StreamingAnalytics` instance for the playback state changes of your player.

Playback state change notification methods

Playback state change	Method	Comments
buffering starts	`notifyBufferStart()`	Indicates the player has started *buffering streaming data and the player is currently **not** playing*. You can call this method when buffering occurs prior to the start of playback as well as when buffering occurs during playback. **It is important to call this method when buffering occurs to ensure time spent buffering is not reported as playing time.**

Playback state change	Method	Comments
buffering ends	`notifyBufferStop()`	Indicates the player has *finished buffering streaming data*. You can call this method whenever you have previously called `notifyBufferStart()` to indicate buffering has finished. If you called `notifyBufferStart()` prior to the start of playback then the `StreamingAnalytics` instance will assume the player is now idle and waiting to start playback. Otherwise, If you called `notifyBufferStart()` during playback then the `StreamingAnalytics` instance will resume the collection of playing time.
playback activates	`notifyPlay()`	Indicates playback has *started* / *resumed after pausing* or *continued after seeking*.
playback pauses	`notifyPause()`	Indicates playback *is paused and the player is currently **not** playing*.

playback ends	`notifyEnd()`	Indicates *playback has ended*. You typically call this method in the following cases:
		▪ Playback naturally reaches the end of the content or advertisement.
		▪ The user interacts with the player, **causing the player to go to an idle state**. This does not necessarily mean the player was playing:
		  ▪ Playback could have been paused.
		  ▪ The player could have been seeking or buffering.
		▪ Playback of the current asset ends because the player needs to change media, for example to load an advertisement for a mid-roll ad break or go back to the content after a mid-roll ad break.
		▪ The player encountered a fatal error during playback, pausing, seeking or buffering and playback cannot continue.

seeking starts	`notifySeekStart()`	Indicates the player has *started seeking*. You typically call this method when the user interacts with the player to make playback resume from a different position on the player *Time Line*. After seeking has finished playback will typically resume from a different position. Please make sure to **call the appropriate API method to make this new position known** to the `StreamingAnalytics` instance as instructed in *Update current Playback Position on page 10*.

2.7 Additional change notifications

Depending on your player's capabilities, the kind of media your player supports and possible playback scenarios, there can also be other changes in the environment which you need to make the `StreamingAnalytics` instance aware of. Relevant situations are described in this section.

2.7.1 Specify *DVR Window Length* for *Live+DVR* streams

In Streaming Tag terminology *Live* refers to the transmission method rather than the media being live recorded. Typically these are multicast, unicast or simulcast deliveries where the player offers the live streams in a way where the user cannot choose what to play: the player will play whatever is being streamed by the media server.

Some players offer DVR ('Digital Video Recorder') capabilities for live streams. In this case the user can seek back and forth in the live stream, typically up to a certain amount of time (for example, 30 minutes or 2 hours back). When the user performs this action, the player will stream what was served on the live stream at that point in time. In Streaming Tag terminology this called *Live+DVR*. These actions by the user can impact metrics collection and need to be addressed in your implementation.

The following definitions are relevant for tagging Live+DVR streams:

Live Edge

The outer edge of the player *Time Line*, typically where a player would start playing a live stream. The user cannot change the *Playback Position forward* when the player is playing from the *Live Edge*. If a player does **not** offer *Live+DVR* capabilities then by definition playback *is always at the live edge* for any live streams.

DVR Window Length

The maximum amount of time the user can go back in time on the live stream. For example: if the player allows the user to go back to what was live streamed *at most* 30 minutes ago, then the *DVR Window Length* is 30 minutes.

DVR Window Offset

The amount of time the current playback position is behind the *Live Edge*. As an example, assume the player has a *DVR Window Length* of 30 minutes and is at the *Live Edge* when this scenario occurs:

1. At the *Live Edge* the *DVR Window Offset* is 0.
2. The user moves the *Playback Position* 12 minutes *backwards* (i.e., the user seeks). When playback continues, the *DVR Window Offset* is now 12 minutes.
3. As playback progresses, the *DVR Window Offset* continues to be 12 minutes.
4. The user moves the *Playback Position forward* by 4 minutes and playback continues, causing the *DVR Window Offset* to now be 8 minutes.

For *Live+DVR* use cases please use the following notification method on the `StreamingAnalytics` instance to inform it of *DVR Window Length* changes.

The `StreamingAnalytics` instance uses this calls to this notification method to identify the current asset as a *Live+DVR* stream to ensure accurate metrics reporting. Please make sure **not** to call this notification method for any live streams where the player does not offer DVR capabilities.

Live+DVR change notification methods

Change	Method	Comments
DVR Live Window Length changes	`setDvrWindowLength( long length )`	Indicates the current *DVR Window Length* is known or has changed. It is expected for this method to be called **before** playback of the live stream starts or resumes — e.g., after pausing, seeking and/or changing to other media such as advertisements — as well as when the *DVR Window Length* changes during playback. The method expects one argument with a **positive** `long` value representing the length in **milliseconds**. For example: ▪ A DVR window length of 30 minutes is represented as `1800000`. ▪ A DVR window length of 2 hours is represented as `7200000`.

Changes to the *DVR Window Offset* are considered playback position changes, for which specific instructions are provided in

Update current Playback Position on this page

2.7.2 Update current *Playback Position*

The `StreamingAnalytics` instance internally automatically calculates the current *Playback Position* from media changes, playback state changes and the progress of natural time while the player is *playing*. For example, when content media playback is interrupted for mid-roll ad breaks, the `StreamingAnalytics` instance automatically uses the content media its last-known position when playback of the content media resumes after the mid-roll ad break.

Although the `StreamingAnalytics` instance can deal with most common use cases, when the following things occur it might be necessary to inform the `StreamingAnalytics` instance where playback will start (or resume) to ensure accurate metrics reporting as the `StreamingAnalytics` instance cannot predict the seeked-to position:

1. When seeking occurs.

2. When the player starts media playback from a non-zero position, or from a position other than the *Live Edge* in case of *Live+DVR* streams.
3. When the player automatically changes the position, for example as the result of playback errors or live streaming behavior.

The are two mechanisms to inform the `StreamingAnalytics` instance of the position where playback will start (or resume), each with their own notification method on the `StreamingAnalytics` instance.

> Please note that the two mechanisms **should not both be used on the same asset** to ensure accurate metrics reporting.

Playback Position change notification methods

Change	Method	Comments
Any non-*Live+DVR* position change	`startFromPosition( long position )`	Indicates the *Playback Position* where playback will start or resume next. Calls to this method will take effect on the next occurrence of playback (not necessarily for the same asset), which can be the start of playback as well as resuming playback after seeking, buffering or changing media. The method expects one argument with a **positive** `long` value representing the *Playback Position* in **milliseconds**. For example: 10 minutes should be provided as `600000`.
DVR Window Offset change	`startFromDvrWindowOffset( long offset )`	Indicates the current *DVR Window Offset* is known or has changed. Calls to this method will take effect on the next occurrence of playback (not necessarily for the same asset). Calling this method will cause the `StreamingAnalytics` instance to identify the current asset as a *Live+DVR* stream to ensure accurate metrics reporting. The method expects one argument with a **positive** `long` value representing the *DVR Window Offset* in **milliseconds**. For example: ▪ A DVR window offset or 0 seconds - i.e., playback is at the *Live Edge* - is represented as `0`. ▪ A DVR window offset of 8 minutes - i.e., playback is 8 minutes in the past from the *Live Edge* - is represented as `480000`.

2.7.3 Add playback rate change notifications

If your player is capable of changing playback rate, then please use the following notification method on the `StreamingAnalytics` instance to indicate each playback rate change and ensure the automatic calculation of playback position and completion metrics are correct.

Playback rate change notification methods

Change	Method	Comments
Playback rate changes	`notifyChangePlaybackRate( float rate )`	The playback rate is expressed as a `Float` value. Example playback rate values are: ▪ normal speed (100%): `1.0` ▪ half speed (50%): `0.5` ▪ double speed (200%): `2.0`

For example, to indicate playback speed has doubled:

```
71.  sa.notifyChangePlaybackRate( 2.0 );
```

Please be aware that the `StreamingAnalytics` instance retains the current playback rate when the current *Asset* changes. If your player resets its playback rate when media changes, then please make sure to include a notification method call to indicate the reset.

Appendix A: Content metadata list

The following table lists the `ContentMetadata.Builder` API methods for specifying metadata values.

Content metadata

V = Video Metrix X = Cross Platform Product Suite C = Cross Media Audience Measurement

<table>
<tr><th>Method</th><th>Required for..</th><th>Optional for...</th><th>Example value</th></tr>
<tr><td rowspan="2">mediaType(value)</td><td>V X C</td><td>—</td><td>ContentType.LONG_FORM_ON_DEMAND</td></tr>
<tr><td colspan="3">The media type is critical for enabling Comscore to distinguish different types of streams. The values are provided with the ContentType object:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SHORT_FORM_ON_DEMAND[A]</td><td rowspan="3">PREMIUM
Content with strong brand equity or brand recognition. Premium content is usually created or produced by media and entertainment companies using professional-grade equipment, talent, and production crews that hold or maintain the rights for distribution and syndication.</td></tr>
<tr><td>LONG_FORM_ON_DEMAND[A]</td></tr>
<tr><td>LIVE</td></tr>
<tr><td>USER_GENERATED_SHORT_FORM_ON_DEMAND[A]</td><td rowspan="3">USER-GENERATED
Content with little-to-no brand equity or brand recognition. User-generated content (UGC) has minimal production value, and is uploaded to the Internet by non-media professionals.</td></tr>
<tr><td>USER_GENERATED_LONG_FORM_ON_DEMAND[A]</td></tr>
<tr><td>USER_GENERATED_LIVE</td></tr>
<tr><td>BUMPER</td><td>BUMPERS[B]
Bumpers — also known as billboards or slates — are static promotional items which usually run before content and usually last less than 5 seconds.</td></tr>
<tr><td>OTHER</td><td>Used if none of the above categories apply.</td></tr>
</table>
[A] Long form video on demand is differentiated from short form video on demand in that long form content always has a content arc with a beginning, middle, and end which in its entirety typically lasts longer than 10 minutes.

[B] Bumpers (billboards, slates) do not have to be tagged. With some implementations tagging of bumpers cannot be avoided. In those cases these values can be used to identify streams as bumpers.</td></tr>
<tr><td rowspan="2">uniqueId(String id)</td><td>V X C</td><td>—</td><td>13784</td></tr>
<tr><td colspan="3">Used in report calculations logic to identify individual content. Provide your internal unique identifier for the content.
Provide value "0" if your media player does not use or have access to unique content identifiers.</td></tr>
<tr><td rowspan="2">length(int length)</td><td>V X C</td><td>—</td><td>1418000 (23 minutes and 58 seconds)</td></tr>
<tr><td colspan="3">A value in milliseconds indicating the length of the individual content (the available amount of content). If your media player or content metadata database reports length values in seconds then please multiply those values by 1000.
If the content length is unknown or cannot be determined then please provide value 0.</td></tr>
<tr><td>dictionaryClassificationC3(String value)
dictionaryClassificationC4(String value)
dictionaryClassificationC6(String value)</td><td>V</td><td>—</td><td>*null</td></tr>
</table>

Method	Required for..	Optional for...	Example value
	These values determine which entity the content will credit to in the Video Metrix dictionary. The values do not have specific pre-defined meanings. You should work with your Comscore account team to establish what these metadata values should be, based on your desired dictionary goals. Provide value `"*null"` for any of the values you do not intend to use.		
`stationTitle( String title )`	V X C	–	▪ `ESPN3` ▪ `BBC2`
	Title of the station or channel for which content was recorded or where content is made available.		
`stationCode( String code )`	–	V X C	`sc132`
	Code of the station or channel for which content was recorded or where content is made available. Can be used for matching purposes (for example when the station titles are multilingual).		
`networkAffiliate( String code )`	–	V X C	▪ `ABC` ▪ `GRIT` ▪ `Escape` ▪ `MeTV`
	Code to identify station affiliation in cases where the same local TV station call sign is affiliated with multiple national TV networks. Expected to be used alongside `stationTitle( String title )` or `stationCode( String code )`.		
`publisherName( String name )`	X C	V	▪ `ABC` ▪ `ESPN` ▪ `CNN`
	Collect the consumer-facing brand name of the media publisher that owns the content.		
`programTitle( String title )`	V X C	–	▪ `Modern Family` ▪ `Harry Potter 7` ▪ `Game 16: Eagles vs Patriots`
	Top level content title (i.e., the name of the overall program, show, or content series). Can be used with `episodeTitle( String title )` to tag TV shows on program and episode level.		
`programId( String id )`	–	V C	`53617155`
	Top level content ID to be used for matching and grouping purposes (for example when the program title appears with multiple variations for the same program). Can be used with `episodeId( String id )` to tag TV shows on program and episode level. *This should not be confused with `uniqueId( String id )` which identifies an individual asset.*		
`episodeTitle( String title )`	V X C	–	▪ `Rash Decisions` ▪ `Season 2 Teaser`
	Sub level content title (i.e., the title of the specific episode). Can be used with `programTitle( String title )` to tag TV shows on program and episode level.		
`episodeId( String id )`	–	X C	`846252126`
	Sub level content ID to be used for matching and grouping purposes (for example when the episode title appears with multiple variations for the same episode of a specific program). Can be used with `programId( String id )` to tag TV shows on program and episode level. *(This should not be confused with `uniqueId( String id )` which identifies an individual asset.)*		
`episodeSeasonNumber( String value )`	X C	–	`05`
	Season number for episodic content. It is recommended to use values with 2 digits, left-padded with `0`. Omit or provide an empty string for non-episodic content.		
`episodeNumber( String value )`	X C	–	▪ `08` ▪ `008`
	Episode number for episodic content. It is recommended to use values with 2 digits — or 3 digits for episodic content with more than 99 episodes in a season — left-padded with `0`.		
`genreName( String name )`	V X C	–	▪ `Comedy` ▪ `Sports` ▪ `Fantasy,Drama`
	Genre description. Multiple values can be provided as a comma-separated string.		
`genreId( String id )`	–	V X C	▪ `243` ▪ `e5a5c` ▪ `165,73`
	Genre ID to be used for matching and grouping purposes (for example when the genres are multilingual). Multiple values can be provided as a comma-separated string.		

Method	Required for..	Optional for...	Example value
`carryTvAdvertisementLoad( Boolean value )`	X	—	`true`
	Use value `true` if the streamed content carries the same advertisement load that was used during the TV airing. Otherwise omit or use value `false`. This metadata helps Comscore differentiate if the stream is carrying the same ad load as TV. Often digital video inventory is clubbed together with TV inventory and is served with the same ad load. The CPM(1) for digital inventory with TV ad load is different from the CPM for any other ad load. If for any reason your backend or workflow requires all media metadata to have values for the same set of metadata, then please make sure you use value `false` for any streamed content which did **not** carry the same advertisement load as during the TV airing.		
`classifyAsCompleteEpisode( Boolean value )`	X	—	`true`
	Use value `true`if the content media is a full episode, rather than an excerpt. Otherwise omit or use value `false`. This metadata helps Comscore identify if the streaming content is episodic, long-form, or premium in nature. It also indicates whether the show or episode will be explicitly broken out in the dictionary. If for any reason your backend or workflow requires all media metadata to have values for the same set of metadata, then please make sure you use value `false` for any streamed media which is **not** a full content episode.		
`dateOfProduction( int year, int month, int day )`	—	C	`2019`, `5`, `14` (May 14, 2019)
	The date on which the content was produced or created.		
`timeOfProduction( int hours, int minutes )`	—	C	`17`, `24` (17:24)
	The time at which the content was produced or created.		
`dateOfTvAiring( int year, int month, int day )`	X C	—	`2019`, `5`, `22` (May 22, 2019)
	The date on which the content aired on TV. This metadata helps Comscore establish monetization windows (live, day +1, day +3, etc.) for any given episode or show. The monetization windows are used to calculate commercial and program ratings.		
`timeOfTvAiring( int hours, int minutes )`	—	X C	`20`, `30` (20:30)
	The time at which the content aired on TV.		
`dateOfDigitalAiring( int year, int month, int day )`	X C	—	`2019`, `5`, `25` (May 25, 2019)
	The date on which the content was made available for streaming consumption. This metadata helps Comscore establish monetization windows (live, day +1, day +3, etc.) for any given episode or show. The monetization windows are used to calculate commercial and program ratings.		
`timeOfDigitalAiring( int hours, int minutes )`	—	X C	`11`, `15` (11:15)
	The time at which the content was made available for streaming consumption.		
`feedType( value )`	X	—	`ContentFeedType.EAST_HD`
	Specify the type of feed provided on the live stream. Intended to be used on live streams using the same feed as was used for the live TV broadcast. Currently only used for implementations in the US. The values are provided with the `ContentFeedType` object:		

Value	Description
`EAST_HD`	Live stream is using the high definition feed used for US eastern live TV broadcast
`WEST_HD`	Live stream is using the high definition feed used for US western live TV broadcast
`EAST_SD`	Live stream is using the standard definition feed used for US eastern live TV broadcast
`WEST_SD`	Live stream is using the standard definition feed used for US western live TV broadcast

`classifyAsAudioStream( Boolean value )`	V X C	—	`true`

(1) CPM — short for 'Cost Per Mille' — is the advertising cost per 1000 impressions.

<table>
<tr><th>Method</th><th>Required for..</th><th>Optional for...</th><th>Example value</th></tr>
<tr><td></td><td colspan="3">Use value <code>true</code> if the content is audio-only, rather than video (with or without audio). Otherwise omit or use value <code>false</code>.
This metadata helps Comscore identify if the streaming content is audio-only in nature.

If for any reason your backend or workflow requires all media metadata to have values for the same set of metadata, then please make sure you use value <code>false</code> for any streamed media which is video (with or without audio).</td></tr>
<tr><td rowspan="2"><code>deliveryMode(value)</code></td><td>–</td><td>V X C</td><td><code>ContentDeliveryMode.ON_DEMAND</code></td></tr>
<tr><td colspan="3">Identifies the content delivery to be on-demand or linear. The values are provided with the <code>ContentDeliveryMode</code> object:
<table><tr><th>Value</th><th>Description</th></tr><tr><td><code>LINEAR</code></td><td>Content delivery was linear</td></tr><tr><td><code>ON_DEMAND</code></td><td>Content delivery was on-demand</td></tr></table></td></tr>
<tr><td rowspan="2"><code>deliverySubscriptionType(value)</code></td><td>–</td><td>V X C</td><td><code>ContentDeliverySubscriptionType.PREMIUM</code></td></tr>
<tr><td colspan="3">Identifies the type of subscription of the user. The values are provided with the <code>ContentDeliverySubscriptionType</code> object:
<table><tr><th colspan="2">Value</th><th>Description</th></tr><tr><td rowspan="2">For live (linear) delivery</td><td><code>TRADITIONAL_MVPD</code></td><td>Traditional Multichannel video programming distributor</td></tr><tr><td><code>VIRTUAL_MVPD</code></td><td>Virtual multichannel video programming distributor</td></tr><tr><td rowspan="4">For on-demand delivery</td><td><code>SUBSCRIPTION</code></td><td>Subscription video on demand</td></tr><tr><td><code>TRANSACTIONAL</code></td><td>Transactional video on demand</td></tr><tr><td><code>ADVERTISING</code></td><td>Advertising video on demand</td></tr><tr><td><code>PREMIUM</code></td><td>Premium video on demand</td></tr></table></td></tr>
<tr><td rowspan="2"><code>deliveryComposition(value)</code></td><td>–</td><td>V X C</td><td><code>ContentDeliveryComposition.CLEAN</code></td></tr>
<tr><td colspan="3">Indicates whether or not ads are delivered as part of the content stream. The values are provided with the <code>ContentDeliveryComposition</code> object:
<table><tr><th>Value</th><th>Description</th></tr><tr><td><code>CLEAN</code></td><td>Advertisements are not delivered as part of the content stream</td></tr><tr><td><code>EMBED</code></td><td>Advertisements are delivered as part of the content stream</td></tr></table></td></tr>
<tr><td rowspan="2"><code>deliveryAdvertisementCapability(value)</code></td><td>–</td><td>V X C</td><td><code>ContentDeliveryAdvertisementCapability.DYNAMIC_LOAD</code></td></tr>
<tr><td colspan="3">Indicate what capability is allowed for advertisement placements. The values are provided with the <code>ContentDeliveryAdvertisementCapability</code> object:
<table><tr><th>Value</th><th>Description</th></tr><tr><td><code>NONE</code></td><td>No advertisement placement allowed</td></tr><tr><td><code>DYNAMIC_LOAD</code></td><td>The allowed advertisement placement capability is dynamic advertisement load</td></tr><tr><td><code>DYNAMIC_REPLACEMENT</code></td><td>The allowed advertisement placement capability is dynamic advertisement replacement</td></tr><tr><td><code>LINEAR_1DAY</code>, <code>LINEAR_2DAY</code>, <code>LINEAR_3DAY</code>, <code>LINEAR_4DAY</code>, <code>LINEAR_5DAY</code>, <code>LINEAR_6DAY</code>, <code>LINEAR_7DAY</code></td><td>The allowed advertisement placement capability is linear ad load for a specific number of days, e.g., <code>LINEAR_3DAY</code> for 3 days</td></tr></table></td></tr>
<tr><td><code>mediaFormat(value)</code></td><td>–</td><td>V X C</td><td><code>ContentMediaFormat.FULL_CONTENT_EPISODE</code></td></tr>
</table>

<table>
<tr><th>Method</th><th>Required for..</th><th>Optional for...</th><th>Example value</th></tr>
<tr><td></td><td colspan="3">Specify the type of content media in more detail. The values are provided with the <code>ContentMediaFormat</code> object:

<table>
<tr><th colspan="2">Value</th><th>Description</th></tr>
<tr><td rowspan="5">For full content</td><td colspan="2">The original content in its entirety (i.e., at least 85%)</td></tr>
<tr><td><code>FULL_CONTENT_EPISODE</code></td><td>Content is a full episode</td></tr>
<tr><td><code>FULL_CONTENT_MOVIE</code></td><td>Content is a full movie</td></tr>
<tr><td><code>FULL_CONTENT_PODCAST</code></td><td>Content is a full podcast</td></tr>
<tr><td><code>FULL_CONTENT_GENERIC</code></td><td>Full content that cannot be identified as a listed format</td></tr>
<tr><td rowspan="5">For partial content</td><td colspan="2">Part of the original content (i.e., less than 85%)</td></tr>
<tr><td><code>PARTIAL_CONTENT_EPISODE</code></td><td>Partial episode</td></tr>
<tr><td><code>PARTIAL_CONTENT_MOVIE</code></td><td>Partial movie</td></tr>
<tr><td><code>PARTIAL_CONTENT_PODCAST</code></td><td>Partial podcast</td></tr>
<tr><td><code>PARTIAL_CONTENT_GENERIC</code></td><td>Partial content that cannot be identified as a listed format</td></tr>
<tr><td rowspan="4">For preview content</td><td colspan="2">A preview or trailer for the original content</td></tr>
<tr><td><code>PREVIEW_EPISODE</code></td><td>Episode preview</td></tr>
<tr><td><code>PREVIEW_MOVIE</code></td><td>Movie preview</td></tr>
<tr><td><code>PREVIEW_GENERIC</code></td><td>Preview for content that cannot be identified as episode or movie</td></tr>
<tr><td rowspan="4">For extra content</td><td colspan="2">Additional content, not part of the original broadcasting</td></tr>
<tr><td><code>EXTRA_EPISODE</code></td><td>Episode extra content</td></tr>
<tr><td><code>EXTRA_MOVIE</code></td><td>Movie extra content</td></tr>
<tr><td><code>EXTRA_GENERIC</code></td><td>Extra content is additional to original content that cannot be identified as episode or movie</td></tr>
</table></td></tr>
<tr><td rowspan="2"><code>distributionModel(value)</code></td><td>–</td><td>V X C</td><td><code>ContentDistributionModel.TV_AND_ONLINE</code></td></tr>
<tr><td colspan="3">Specify where the content was distributed. The values are provided with the <code>ContentDistributionModel</code> object:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>TV_AND_ONLINE</code></td><td>Content is distributed on TV and online</td></tr>
<tr><td><code>EXCLUSIVELY_ONLINE</code></td><td>Content is distributed exclusively online</td></tr>
</table></td></tr>
<tr><td rowspan="2"><code>playlistTitle(String title)</code></td><td>–</td><td>C</td><td><code>"Modern Family Season 2"</code></td></tr>
<tr><td colspan="3">Can be used if the player offers the media as part of a playlist. Specify an identifier (title, etc.) for the playlist. For example, the TV Show title for a playlist which contains all episodes from a specific TV show.</td></tr>
<tr><td rowspan="2"><code>totalSegments(int total)</code></td><td>–</td><td>V C</td><td><code>3</code></td></tr>
<tr><td colspan="3">Indicates the total number of segments of the content, which is one more than the number of mid-roll ad breaks. For example, 1 segment means no mid-roll ad breaks while 3 segments means 2 mid-roll ad breaks.
Provide value <code>0</code> if the total number of segments of the content cannot be determined.</td></tr>
<tr><td rowspan="2"><code>clipUrl(String url)</code></td><td>–</td><td>V C</td><td><code>http://streaming.example.com/asset/13784</code></td></tr>
<tr><td colspan="3">The URL (or path/filename) of the content stream.</td></tr>
<tr><td rowspan="2"><code>videoDimensions(int pixelsWide, int pixelsHigh)</code></td><td>–</td><td>C</td><td><code>1280, 720</code></td></tr>
<tr><td colspan="3">Content video width and height in pixels.</td></tr>
</table>

Method	Required for..	Optional for...	Example value

`customLabels( Map labels )`	–	V X C	`HashMap<String,String> labels = new HashMap<String,String>();`
			`labels.put("name1", "value1");`
			`labels.put("name2", "value2");`

	Can be used to specify a collection of custom metadata name/value pairs.		

Appendix B: Advertisement metadata list

The following table lists the `AdvertisementMetadata.Builder` API methods for specifying metadata values.

Advertisement metadata

V = Video Metrix X = Cross Platform Product Suite C = Cross Media Audience Measurement

<table>
<tr><th>Method</th><th>Required for..</th><th>Optional for...</th><th>Example value</th></tr>
<tr><td rowspan="2">mediaType(value)</td><td>V X C</td><td>–</td><td>AdvertisementType.ON_DEMAND_PRE_ROLL</td></tr>
<tr><td colspan="3">The media type is critical for enabling Comscore to distinguish different types of streams. The values are provided with the AdvertisementType object:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>ON_DEMAND_PRE_ROLL</td><td rowspan="3">LINEAR - VIDEO ON DEMAND
Linear advertisements delivered into a media player and presented before, in the middle of, or after video content is consumed by the user. The advertisement completely takes over the full view of the media player.</td></tr>
<tr><td>ON_DEMAND_MID_ROLL</td></tr>
<tr><td>ON_DEMAND_POST_ROLL</td></tr>
<tr><td>LIVE</td><td>LINEAR - LIVE
Linear advertisements delivered before, in the middle of, or after a live stream of content. The advertisement completely takes over the full view of the media player.</td></tr>
<tr><td>BRANDED_ON_DEMAND_PRE_ROLL</td><td rowspan="5">BRANDED ENTERTAINMENT
Media that a user may intentionally view (like content), or it may be served to a user during an ad break (like an advertisement).</td></tr>
<tr><td>BRANDED_ON_DEMAND_MID_ROLL</td></tr>
<tr><td>BRANDED_ON_DEMAND_POST_ROLL</td></tr>
<tr><td>BRANDED_AS_CONTENT</td></tr>
<tr><td>BRANDED_DURING_LIVE</td></tr>
<tr><td>OTHER</td><td>OTHER
Used if none of the above categories apply.</td></tr>
</table></td></tr>
<tr><td rowspan="2">relatedContentMetadata(contentMetadataObject)</td><td>V X C</td><td>–</td><td>cm</td></tr>
<tr><td colspan="3">Specify the ContentMetadata of the content which the advertisement is served for. Omit for cases where player is not aware which content the advertisement is playing for.</td></tr>
<tr><td rowspan="2">uniqueId(String id)</td><td>–</td><td>C</td><td>"332584"</td></tr>
<tr><td colspan="3">Provide a unique identifier of the advertisement. The identifier is expected to different for different advertisements (i.e., to distinguish one creative from another).
Provide value "0" if your media player does not use or have access to unique content identifiers.</td></tr>
<tr><td rowspan="2">length(int length)</td><td>X</td><td>V C</td><td>27000 (27 seconds)</td></tr>
<tr><td colspan="3">A value in milliseconds indicating the length of the individual advertisement. If your media player or advertisement metadata reports length values in seconds then please multiply those values by 1000.
If the advertisement length is unknown or cannot be determined then please provide value 0.</td></tr>
<tr><td rowspan="2">deliveryType(value)</td><td>–</td><td>X C</td><td>AdvertisementDeliveryType.NATIONAL</td></tr>
<tr><td colspan="3">Specify the mechanism use to deliver an advertisement. The values are provided with the AdvertisementDeliveryType object:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>NATIONAL</td><td>The advertisement is delivered nationally</td></tr>
<tr><td>LOCAL</td><td>The advertisement is delivered locally</td></tr>
<tr><td>SYNDICATION</td><td>The advertisement is delivered for syndication</td></tr>
</table></td></tr>
<tr><td>owner(value)</td><td>–</td><td>X C</td><td>AdvertisementOwner.DISTRIBUTOR</td></tr>
</table>

<table>
<tr><th>Method</th><th>Required for..</th><th>Optional for...</th><th>Example value</th></tr>
<tr><td></td><td colspan="3">Specify who is monetizing the advertisement.The values are provided with the <code>AdvertisementOwner</code> object:
<table><tr><th>Value</th><th>Description</th></tr><tr><td><code>DISTRIBUTOR</code></td><td>Advertisement is monetized by distributor (i.e., the party reflected by the <code>publisherName(String name)</code> metadata)</td></tr><tr><td><code>ORIGINATOR</code></td><td>Advertisement is monetized by originator (i.e., the party reflected by the <code>stationTitle(String title)</code> or <code>stationCode(String code)</code> metadata)</td></tr><tr><td><code>MULTIPLE</code></td><td>Advertisement is monetized by multiple owners</td></tr><tr><td><code>NONE</code></td><td>Advertisement is not owned</td></tr></table></td></tr>
<tr><td rowspan="2"><code>classifyAsAudioStream(Boolean value)</code></td><td>V X C</td><td>–</td><td><code>true</code></td></tr>
<tr><td colspan="3">Use value <code>true</code> if the advertisement is audio-only, rather than video (with or without audio). Otherwise omit or use value <code>false</code>.
This metadata helps Comscore identify if the streaming advertisement is audio-only in nature.

If for any reason your backend or workflow requires all media metadata to have values for the same set of metadata, then please make sure you use value <code>false</code> for any streamed media which is video (with or without audio).</td></tr>
<tr><td rowspan="2"><code>serverCampaignId(String id)</code></td><td>–</td><td>X C</td><td><code>"5237817254"</code></td></tr>
<tr><td colspan="3">Provide an ID for the advertisement campaign being delivered.</td></tr>
<tr><td rowspan="2"><code>placementId(String id)</code></td><td>–</td><td>X C</td><td><code>"867225"</code></td></tr>
<tr><td colspan="3">Provide an ID for the placement the advertisement campaign is being delivered to.</td></tr>
<tr><td rowspan="2"><code>siteId(String id)</code></td><td>–</td><td>X C</td><td><code>"3445"</code></td></tr>
<tr><td colspan="3">Provide an ID for the site the advertisement campaign is being delivered to.</td></tr>
<tr><td rowspan="2"><code>server(String name)</code></td><td>–</td><td>X C</td><td><code>"Freewheel"</code></td></tr>
<tr><td colspan="3">Provide a name for the advertising server/provider.</td></tr>
<tr><td rowspan="2"><code>title(String title)</code></td><td>–</td><td>X C</td><td><code>Summer sale 2019</code></td></tr>
<tr><td colspan="3">Provide a title for the advertisement (i.e., the name of the campaign or creative).</td></tr>
<tr><td rowspan="2"><code>callToActionUrl(String url)</code></td><td>–</td><td>C</td><td><code>"http://example.com/landing_page"</code></td></tr>
<tr><td colspan="3">Provide the URL which will be loaded when the advertisement is clicked on.</td></tr>
<tr><td rowspan="2"><code>clipUrl(String url)</code></td><td>–</td><td>C</td><td><code>http://streaming.example.com/asset/13784</code></td></tr>
<tr><td colspan="3">The URL (or path/filename) of the advertisement stream.</td></tr>
<tr><td rowspan="2"><code>videoDimensions(int pixelsWide, int pixelsHigh)</code></td><td>–</td><td>C</td><td><code>1280</code>, <code>720</code></td></tr>
<tr><td colspan="3">Advertisement video width and height in pixels.</td></tr>
<tr><td rowspan="2"><code>customLabels(Map labels)</code></td><td>–</td><td>V X C</td><td><code>HashMap<String,String> labels = new HashMap<String,String>();
labels.put("name1", "value1");
labels.put("name2", "value2");</code></td></tr>
<tr><td colspan="3">Can be used to specify a collection of custom metadata name/value pairs.</td></tr>
</table>

Appendix C: Content metadata example values

There are different types of video content out there on the internet and each type has certain nuances about how it should be tagged in order to be reported correctly in Comscore's Audience measurement products. This section will guide you how to populate the video metadata parameters for the most common types of content available on the internet.

Content metadata examples per type of content

Metadata	TV Show Episode	TV Show Trailer	Live Sports Content	Sports Highlight Clip	Movie	Movie Trailer	Online News Content	Music Video
Station Title — `stationTitle( String title )`	`Hulu`	`YouTube`	`ESPN3`	`YouTube`	`Hulu`	`YouTube`	`Huffington Post`	`VEVO`
Publisher Brand Name — `publisherName( String name )`	`ABC`	`ABC`	`ESPN`	`NFL`	`Warner Bros.`	`Warner Bros.`	`Huffington Post`	`VEVO`
Program Title — `programTitle( String title )`	`Modern Family`	`Modern Family`	`Game 16: Eagles vs Patriots`	`Game 16: Eagles vs Patriots`	`Harry Potter 7`	`Harry Potter 7`	`Huff Post Live`	`Taylor Swift`
Episode Title — `episodeTitle( String title )`	`Rash Decisions`	`Season 2 Teaser`	`*null`	`*null`	`*null`	`Harry Potter 7 Trailer #3`	`All is not Well in Hillaryland`	`Wildest Dreams`
Episode Season Number — `episodeSeasonNumber( String value )`	`1`	`*null`	`*null`	`*null`	`*null`	`*null`	`*null`	`*null`
Episode Number — `episodeNumber( String value )`	`2`	`*null`	`*null`	`*null`	`*null`	`*null`	`*null`	`*null`
Genre — `genreName( String name )`	`Comedy`	`Comedy`	`Sports`	`Sports`	`Fantasy, Drama`	`Fantasy, Drama`	`News`	`Music`
Complete Episode — `classifyAsCompleteEpisode( Boolean value )`	`1`	`0`	`1`	`0`	`1`	`0`	`0`	`0`

A list of suggested Genre values is provided below:

- Action / Adventure
- Adult
- Animation
- Awards
- Comedy
- Food
- Documentary
- Drama
- Educational
- Fantasy
- Foreign Language
- Game Show
- Holiday
- Home & Garden / Home Improvement
- Home Shopping
- Kids
- Lifestyle
- Movies
- Music
- News
- Paid Programming
- Politics / Public Affairs
- Reality
- Religious
- Science Fiction
- Soap Opera
- Sports
- Talk
- Thriller / Horror
- Travel
- Variety

Appendix D: Update an existing implementation

Updating within the same library major version typically are drop-in replacements. When *upgrading* to a newer major version some code changes might be required as major versions usually include API changes.

Your development environment will typically point out when your code uses API methods which no longer exist or have a different signature in newer library. This appendix aims to give an impression of what can be involved in an upgrade.

> It could be that some of the library classes, API methods or method arguments mentioned in this appendix do not appear in your implementation. If your implementation contains elements which are not mentioned in these migration instructions then please contact your Comscore account team or implementation support team for additional instructions.

There are a few Comscore library major versions in circulation. You can determine which one you have implemented from the required configuration code statements.

Determine library version

Major Version	Appearance / Characteristics

5	The configuration code statements look like:
	`PublisherConfiguration myPublisherConfig = new PublisherConfiguration.Builder()`
	`.publisherId( "1234567" ) // Will mention your Publisher ID.`
	`.publisherSecret( "7b94840eb66b17e61c3d2f909c3a1163" ) // Will mention your Publisher Secret.`
	`.build();`
	`Analytics.getConfiguration().addClient( myPublisherConfig );`

2 and 3	The configuration code statements look like:
	`comScore.setCustomerC2( "1234567" ); // Will mention your Publisher ID.`
	`comScore.setPublisherSecret( "7b94840eb66b17e61c3d2f909c3a1163" ); // Will mention your Publisher Secret.`

Once you have determined which Comscore library major version you have implemented please first ensure you have updated the integration of the Comscore library in your application project. This means the code statements you have just located in order to determine the library major version will be updated first.

Next, you will need to determine the type of your current Streaming Tag implementation in order to know which migration steps to follow.

Determine type of Streaming Tag implementation

Appearance / Characteristics	Major Version	Implementation Type
Your implementation uses `StreamingAnalytics` object instances	5	**'Standard' Streaming Tag**
Your implementation uses `ReducedRequirementsStreamingAnalytics` object instances		**Reduced Requirements Streaming Tag**
Your implementation uses `StreamSense` object instances	2 and 3	**'Standard' Streaming Tag**
Your implementation uses `StreamingTag` object instances		**Reduced Requirements Streaming Tag**

Migrate 'Standard' Streaming Tag from major version 5 to 6

1. Remove existing import statements which refer to package `com.comscore.streaming`.
2. Assuming you use `sa` to reference the `StreamingAnalytics` object instance, replace the following method calls to account for API changes.

Existing code	Migrated code
111. `sa.getPlaybackSession().setAsset(metadata);`	111. `/* Use AdvertisementMetadata if the asset is an advertisement.` 112. `* You can determine if the asset is an advertisement from` 113. `* the presence and value of ns_st_ad on the metadata argument.` 114. `* If ns_st_ad is present with a value that is` 115. `* not null, empty string, "0" or 0, then the asset is an advertisement.` 116. `*/` 117. `ContentMetadata cm = new ContentMetadata.Builder()` 118. `.customLabels( metadata )` 119. `.build()` 120. `;` 121. `sa.setMetadata(cm);`
121. `sa.notifyBufferStart( position );`	121. `sa.startFromPosition( position );` 122. `sa.notifyBufferStart();`
125. `sa.notifyBufferStop( position );`	125. `sa.startFromPosition( position );` 126. `sa.notifyBufferStop();`
131. `sa.notifyPlay( position );`	131. `sa.startFromPosition( position );` 132. `sa.notifyPlay();`
141. `sa.notifyPause( position );`	141. `sa.notifyPause();`
151. `sa.notifySeekStart( position );`	151. `sa.notifySeekStart();`
161. `sa.notifyEnd( position );`	161. `sa.notifyEnd();`
171. `sa.setDVRWindowLength( length );`	171. `sa.setDvrWindowLength( length );`
175. `sa.setDVRWindowOffset( offset );`	175. `sa.startFromDvrWindowOffset( offset );`

2. Ensure your IDE has included the relevant imports for the classes which are used in the migrated code statements and that the imports for classes which no longer exist have been removed from your code.

Migrate Reduced Requirements Streaming Tag from major version 5 to 6

1. Remove existing import statements which refer to package `com.comscore.streaming`.
2. Replace occurrences of class name `ReducedRequirementsStreamingAnalytics` with `StreamingAnalytics`.

3. Replace occurrences of class name `AdType` with `AdvertisementType`.
4. Assuming you use `sa` to reference the `StreamingAnalytics` object instance, replace the following method calls to account for API changes.

Existing code	Migrated code

`sa.playVideoContentPart( metadata, contentType );`	`ContentMetadata cm = new ContentMetadata.Builder()`
	`  .mediaType( contentType )`
	`  .customLabels( metadata )`
	`  .build()`
	`  ;`
	`sa.setMetadata( cm );`
	`sa.notifyPlay();`

`sa.playAudioContentPart( metadata, contentType );`	`ContentMetadata cm = new ContentMetadata.Builder()`
	`  .mediaType( contentType )`
	`  .classifyAsAudioStream( true )`
	`  .customLabels( metadata )`
	`  .build()`
	`  ;`
	`sa.setMetadata( cm );`
	`sa.notifyPlay();`

`sa.playVideoAdvertisement( metadata, advertisementType );`	`AdvertisementMetadata am = new AdvertisementMetadata.Builder()`
	`  .mediaType( advertisementType )`
	`  .customLabels( metadata )`
	`  .build()`
	`  ;`
	`sa.setMetadata( am );`
	`sa.notifyPlay();`

`sa.playAudioAdvertisement( metadata, advertisementType );`	`AdvertisementMetadata am = new AdvertisementMetadata.Builder()`
	`  .mediaType( advertisementType )`
	`  .classifyAsAudioStream( true )`
	`  .customLabels( metadata )`
	`  .build()`
	`  ;`
	`sa.setMetadata( am );`
	`sa.notifyPlay();`

`sa.stop();`	`sa.notifyPause();`

5. Ensure your IDE has included the relevant imports for the classes which are used in the migrated code statements and that the imports for classes which no longer exist have been removed from your code.

Migrate ‘Standard’ Streaming Tag from major version 2 or 3 to 6

1. Remove existing import statements which refer to package `com.comscore.streaming`.
2. Replace occurrences of class name `StreamSense` with `StreamingAnalytics`.
3. Assuming you use `sa` to reference the `StreamingAnalytics` object instance, replace the following method calls to account for API changes.

Existing code	Migrated code
121. `sa.setClip( labels )`	121. `/* Use AdvertisementMetadata if the asset is an advertisement.` 122. `* You can determine if the asset is an advertisement from` 123. `* the presence and value of ns_st_ad on the labels argument.` 124. `* If ns_st_ad is present with a value that is` 125. `* not null, empty string, "0" or 0, then the asset is an advertisement.` 126. `*/` 127. `ContentMetadata cm = new ContentMetadata.Builder()` 128. `.customLabels( labels )` 129. `.build()` 130. `;` 131. `sa.setMetadata( cm );`
131. `sa.notify( StreamSenseEventType.BUFFER, position )`	131. `sa.startFromPosition( position );` 132. `sa.notifyBufferStart();`
141. `sa.notify( StreamSenseEventType.PLAY, position )`	141. `sa.startFromPosition( position );` 142. `sa.notifyPlay();`
151. `sa.notify( StreamSenseEventType.PAUSE, position )`	151. `sa.notifyPause();`
161. `sa.notify( StreamSenseEventType.END, position )`	161. `sa.notifyEnd();`

4. Ensure your IDE has included the relevant imports for the classes which are used in the migrated code statements and that the imports for classes which no longer exist have been removed from your code.

Migrate Reduced Requirements Streaming Tag from major version 2 or 3 to 6

1. Remove existing import statements which refer to package `com.comscore.streaming`.
2. Replace occurrences of class name `StreamingTag` with `StreamingAnalytics`
3. Replace the following enum occurrences in your code.

Old name	New name
`ContentType.LongFormOnDemand`	`ContentType.LONG_FORM_ON_DEMAND`
`ContentType.ShortFormOnDemand`	`ContentType.SHORT_FORM_ON_DEMAND`
`ContentType.Live`	`ContentType.LIVE`
`ContentType.UserGeneratedLongFormOnDemand`	`ContentType.USER_GENERATED_LONG_FORM_ON_DEMAND`
`ContentType.UserGeneratedShortFormOnDemand`	`ContentType.USER_GENERATED_SHORT_FORM_ON_DEMAND`
`ContentType.UserGeneratedLive`	`ContentType.USER_GENERATED_LIVE`
`ContentType.Bumper`	`ContentType.BUMPER`
`ContentType.Other`	`ContentType.OTHER`
`AdType.LinearOnDemandPreRoll`	`AdvertisementType.ON_DEMAND_PRE_ROLL`
`AdType.LinearOnDemandMidRoll`	`AdvertisementType.ON_DEMAND_MID_ROLL`
`AdType.LinearOnDemandPostRoll`	`AdvertisementType.ON_DEMAND_POST_ROLL`
`AdType.LinearLive`	`AdvertisementType.LIVE`
`AdType.BrandedOnDemandPreRoll`	`AdvertisementType.BRANDED_ON_DEMAND_PRE_ROLL`

Old name	New name
`AdType.BrandedOnDemandMidRoll`	`AdvertisementType.BRANDED_ON_DEMAND_MID_ROLL`
`AdType.BrandedOnDemandPostRoll`	`AdvertisementType.BRANDED_ON_DEMAND_POST_ROLL`
`AdType.BrandedOnDemandContent`	`AdvertisementType.BRANDED_ON_DEMAND_AS_CONTENT`
`AdType.BrandedOnDemandLive`	`AdvertisementType.BRANDED_ON_DEMAND_DURING_LIVE`
`AdType.Other`	`AdvertisementType.OTHER`

4. Assuming you use `sa` to reference the `StreamingAnalytics` object instance, replace the following method calls to account for API changes.

Existing code	Migrated code

`sa.playVideoContentPart( metadata, contentType );`	`ContentMetadata cm = new ContentMetadata.Builder()`
	`.mediaType( contentType )`
	`.customLabels( metadata )`
	`.build()`
	`;`
	`sa.setMetadata( cm );`
	`sa.notifyPlay();`

`sa.playAudioContentPart( metadata, contentType );`	`ContentMetadata cm = new ContentMetadata.Builder()`
	`.mediaType(contentType)`
	`.classifyAsAudioStream( true )`
	`.customLabels( metadata )`
	`.build()`
	`;`
	`sa.setMetadata(cm);`
	`sa.notifyPlay();`

`sa.playVideoAdvertisement( metadata, advertisementType );`	`AdvertisementMetadata am = new AdvertisementMetadata.Builder()`
	`.mediaType( advertisementType )`
	`.customLabels( metadata )`
	`.build()`
	`;`
	`sa.setMetadata( am );`
	`sa.notifyPlay();`

`sa.playAudioAdvertisement(metadata, advertisementType);`	`AdvertisementMetadata am = new AdvertisementMetadata.Builder()`
	`.mediaType( advertisementType )`
	`.classifyAsAudioStream( true )`
	`.customLabels( metadata )`
	`.build()`
	`;`
	`sa.setMetadata( am );`
	`sa.notifyPlay();`

`sa.stop();`	`sa.notifyPause();`

5. Ensure your IDE has included the relevant imports for the classes which are used in the migrated code statements and that the imports for classes which no longer exist have been removed from your code.