

Adobe® Primetime

Offline Packager Getting Started

Contents

| | |
|---|----------|
| Primetime Offline Packager Getting Started..... | 4 |
| What's new in Offline Packager..... | 4 |
| Overview of Offline Packager..... | 5 |
| System requirements..... | 7 |
| Work with Offline Packager..... | 7 |
| Creating output using command-line arguments with a configuration file..... | 7 |
| Validating the configuration..... | 8 |
| Getting help..... | 8 |
| DASH profiles | 8 |
| Multi-bit rate packaging | 9 |
| Packaging multi-bit rate for DASH..... | 9 |
| Packaging multi-bit rate for HLS..... | 9 |
| Packaging multi-bit rate for HDS..... | 10 |
| Manage various tracks in a stream..... | 10 |
| Find the track information..... | 11 |
| PlaylistCreator tool..... | 11 |
| Content protection..... | 17 |
| Protect content using AAXS DRM..... | 17 |
| External CEK support to package HLS content with AAXS DRM..... | 18 |
| Protect content using Widevine DRM for DASH..... | 19 |
| Protect content using Sample-AES encryption..... | 20 |
| Protect content using FairPlay DRM..... | 20 |
| Protect content using multi-DRM for HLS (AAXS and FairPlay)..... | 21 |
| Protect DASH content using AAXS and Playready DRM..... | 21 |
| Protect content using multi-DRM for DASH (AAXS, Playready, and Widevine)..... | 23 |
| DRM/FAXS impact..... | 23 |
| Protect HLS content using AES-128 encryption..... | 24 |
| Enable PHDS and PHLS..... | 25 |
| ms_unencrypted support for HLS and HDS..... | 26 |
| Custom IV support..... | 27 |

| | |
|---|----|
| Whitelist an application for Adobe Access..... | 27 |
| Enable app whitelisting for HLS..... | 27 |
| Enable SWF verification for HDS..... | 28 |
| Refresh the license for HDS and HLS..... | 28 |
| Use HSM to store packager credentials..... | 29 |
| Generate a trick play stream..... | 30 |
| Closed Captioning (CC) support..... | 31 |
| Packaging WebVTT files for HLS delivery..... | 32 |
| Providing Closed Captions for MPEG-DASH delivery..... | 33 |
| Still Image support for audio-only stream..... | 33 |
| Inserting advertisement cues | 34 |
| Advertisement cues for HLS..... | 34 |
| Advertisement cues for HDS..... | 35 |
| Advertisement cues for MPEG-DASH..... | 36 |
| Insert the embedded cue information..... | 36 |
| Ad cues in command line..... | 37 |
| Align Ad Cues with keyframes..... | 37 |
| SCTE cues..... | 37 |
| Cue Signal Format for manifest files..... | 37 |
| Supported SCTE-35 splice_insert() messages..... | 38 |
| Configuration parameters..... | 38 |

Primestime Offline Packager Getting Started

What's new in Offline Packager

The following releases of Offline Packager offers the listed new features.

What's new in Offline Packager version 2.4

Offline Packager 2.4 offers the following features and enhancements:

- [SRT to VTT support](#)
- [External CEK support for HLS AAXS](#)

What's new in Offline Packager version 2.3.1

Offline Packager 2.3.1 offers the following features and enhancements:

- Enable the [On-Demand Profile](#) for DASH content
- Support for `validate` option for the [PlaylistCreator tool](#) to validate the configuration file.
- A few bug fixes for multi-DRM scenarios

What's new in Offline Packager version 2.3

Offline Packager 2.3 offers the following features:

- [Widevine DRM support for DASH](#)
- [Multi-DRM support for DASH](#)

What's new in Offline Packager, Version 2.2

Offline Packager 2.2 offers the following two features:

- [Sample-AES encryption](#) and [FairPlay support](#) for HLS
- [Multi-DRM support](#) for HLS streams

What's new in Offline Packager, Version 2.1

Offline Packager 2.1 offers the following features:

- [DRM support for DASH](#)

What's new in Offline Packager, Version 2.0

Offline Packager 2.0 offers MPEG-DASH support for the following features:

- [Generate a trick play stream](#)
- [Providing Closed Captions for MPEG-DASH delivery](#)
- Alternate audio. See [Manage various tracks in a stream](#).
- [Advertisement cues for MPEG-DASH](#)
- Playlist creation for DASH. See [PlaylistCreator tool](#).

What's new in Offline Packager, Version 1.4

Offline Packager 1.4 offers the following features:

- Custom CEK and [Custom IV support](#) for AAXS workflow
- [Still Image support for audio-only stream](#)
- [Custom IV support](#)
- Enhancements in the [PlaylistCreator tool](#)

What's new in Offline Packager, Version 1.3

Offline Packager 1.3 offers the following features:

- [Using HSM to store packager credentials](#)
- [PlaylistCreator tool](#)
- [ms_unencrypted support for HLS](#)
- [Ad cues in command line](#)
- [Align Ad Cues with keyframes](#)

What's new in Offline Packager Version 1.2

Offline Packager 1.2 offers the following features:

- [SCTE cues](#)
- [Packaging WebVTT files for HLS delivery](#)
- [DRM/FAXS impact](#)

Overview of Offline Packager

The Offline Packager is a command-line utility that packages the content available on the disk.

Broadcasters have a need to stream VOD libraries to their subscribers. Adobe Primetime Offline Packager packages MP4 and TS files for Dynamic Adaptive Streaming over HTTP (MPEG DASH), HTTP Live Streaming (HLS), or HTTP Dynamic Streaming (HDS). It only packages DASH, HLS, and HDS streams but does not transcode them. The Offline Packager requires multi-bit rate source content but does not generate MBR renditions.

The Offline Packager supports content protection using the supported methods. See [Content Protection](#) for all the supported methods.

Table 1: Offline Packager's input and output for different protocols

| Streaming protocol | Input file | Output | Content protection |
|--------------------|------------|--|---|
| DASH | MP4 or TS | Manifest file .mpd (for single-stream playlist, both the MBR and single-stream MPDs are generated depending on whether an MBR or a single stream is being packaged). It is an XML-based playlist file that contains information about codec, resolution, and the | When content protection is enabled and AAXS DRM is used, the output is a .pssh file. It contains DRM specific contents of Protection System Specific Header (PSSH) Box. It is not generated when the option <code>inline_drm</code> is specified. |

| Streaming protocol | Input file | Output | Content protection |
|--------------------|------------|--|---|
| | | availability of multi-bitrate files. DASH audio/video/vtt segments and corresponding initialization segments, as listed in the MPD file. Offline packager produces output conforming to <code>urn:mpeg:dash:profile:isoff-live:2011</code> profile with type <code>static</code> by default. To package the content conforming to an on-demand profile, use the <code>on_demand_dash_profile</code> configuration parameter. | |
| HLS | MP4 or TS | m3u8 file (for single-stream playlist, both the MBR and single-stream m3u8s are generated depending on whether an MBR or a single stream is being packaged). It contains information about codec, resolution, and the availability of multi-bitrate files. It is a text-based playlist file. TS segments are listed in the m3u8 file. | If content protection is enabled, .drm file. |
| HDS | MP4 or TS | <ul style="list-style-type: none"> Fragments are files containing packaged audio and video that do not have a file name extension and are listed in the f4m manifest. F4M is a Flash Media manifest file that is an XML-based playlist that contains information about codec, resolution, and the availability of multi-bitrate files. | For Adobe Access, <code>drmmeta</code> is DRM additional header file. It contains additional header information about an encrypted file. It is an external file, unless the <code>inline_drm</code> tag is specified. |

System requirements

- Linux/CentOS Enterprise 6.3, 64-bit only
- Windows/Windows 2008 Server
- JDK version 1.8 or later



Tip: Local administrator privileges are required on the machine.

Work with Offline Packager

There are two ways you can run Offline Packager.

- Specifying various parameters as command-line arguments.

For example:

```
java -jar OfflinePackager.jar -in_path "D:\artbeats.mp4" -out_type hds -out_path "C:\hds"
```

- Specifying various parameters in a configuration file.

For example:

```
java -jar OfflinePackager.jar -conf_path absolute_path_of_configuration_file
```

The following is a sample configuration file:

```
<config>
<in_path>D:\artbeats.mp4</in_path>
<lang>eng</lang>
<out_type>hls</out_type>
<out_path>C:\hls</out_path>
</config>
```

Creating output using command-line arguments with a configuration file

You can use command-line parameters along with a configuration file. If a parameter is present in both command-line and in the configuration file, the command-line parameter takes precedence.

As a best practice, create a configuration file that contains the common options that you want to use for generating the outputs. Then, create the output by providing specific options as a command-line argument.

For example, you can have common settings in a configuration file and package the content for HDS, HLS, and DASH by running the following commands:

- For HDS output:

```
java -jar OfflinePackager.jar -conf_path config.xml -out_type hds -out_path
/out/hds
```

- For HLS output:

```
java -jar OfflinePackager.jar -conf_path config.xml -out_type hls -out_path
/out/hls
```

- For DASH output:

```
java -jar OfflinePackager.jar -conf_path config.xml -out_type dash -out_path /out/dash
```

Validating the configuration

You can run a command to validate the configuration.

To validate the configuration file, run the following command:

```
java -jar OfflinePackager.jar -conf_path config.xml -validate
```

If you specify the `-validate` parameter, only validation is performed; no packaging is done.

Getting help

To get help while working with Offline Packager, run the following command:

```
java -jar OfflinePackager.jar -help
```

DASH profiles

Offline packager supports the following DASH profiles for packaging.

Live profile for VoD

The ISO Live profile is identified by the URN `urn:mpeg:dash:profile:isoff-live:2011`. Even though this profile is optimized for Live services, it can be used to distribute on-demand content; in such a case the `MPD@Type` attribute is set to `static`. To comply with this profile, the content is packaged as short segments containing one or more movie fragments of ISO file format. Each movie fragment may be requested using a template generated URL. For on-demand content, all segments are available for delivery immediately and independent HTTP requests can be made for each segment. Offline packager generates content conforming to this profile by default.

On-demand profile for VoD

This profile provides basic support for On-demand content. Each representation is available as a single segment. The sub-segments start with Stream Access Points (IDR or key frames). The sub-segments are also aligned across representations, which enables seamless switching. The addressing for on-demand profile is done via a segment index, which maps the presentation time to the corresponding byte range within the segment. The segment index is used by the client to make byte range requests for the sub-segments.

The ISO On-demand profile is identified by the URN `'urn:mpeg:dash:profile:isoff-on-demand:2011'`.

Packaging for on-demand profile

To package content using on-demand profile, use the following command line.

```
java -jar OfflinePackager.jar --in_path sample.mp4 --out_type dash --out_path dashoutput --on_demand_dash_profile
```

If `on_demand_dash_profile` parameter is not specified, the default profile used is `isoff-live` with type `static`.

Multi-bit rate packaging

Multi-bit rate packaging allows a player to switch to a different rendition for optimum playback experience depending on the network traffic and bandwidth available. Using the Offline Packager, you can package media files of different bit rates. When packaging multi-bit rate encrypted content, if `key_out_path` is configured, the keys are written inside individual bitrate folders at `key_out_path` path.



Note: Multi-bit-rate packaging is deprecated but supported in version 2.0. It is recommended to use the [PlaylistCreator tool](#) for creating set-level manifest files.

Packaging multi-bit rate for DASH

1. To enable multi-bit rate packaging, encode a media file at different bit rates.
In this example, we have two files. They are encoded in 550 kbps and 1000 kbps respectively.
2. Place the input files in different directories. The `in_path` argument is relative to the command line location, so use an absolute path if necessary.
In the following example, the 550 kbps file is in the `mbr_hds/500` directory and the 1000 kbps file is in the `mbr_hds/1000` directory.

3. Create a configuration file to package the files by specifying the bit rates. Save the configuration file.

```
<config>
<in_path>384x288_Baseline3.0.mp4,640x360_Baseline3.0.mp4</in_path>
<bit_rates>550,1000</bit_rates>
<top_level_prefix>abr_set</top_level_prefix>
<out_type>dash</out_type>
<out_path>mbr_dash</out_path>
</config>
```

4. Run the Offline Packager.
It also creates a set level manifest file (`abr_set.mpd`) in the `out_path` tag specified. Unlike for HDS and HLS, set level MPD does not refer to stream-level manifest files, but instead it is a merged representation of all stream level MPD files and contains information about all stream-level segments.

Packaging multi-bit rate for HLS

1. To enable multi-bit rate packaging, encode a media file at different bit rates.
In this example, we have two files. They are encoded in 550 kbps and 1000 kbps respectively.
2. Place the input files in different directories. The `in_path` argument is relative to the command line location, so use an absolute path if necessary.
In the following example, the 550 kbps file is in the `mbr_hls/500` directory and the 1000 kbps file is in the `mbr_hls/1000` directory.

3. Create a configuration file to package the files by specifying the bit rates. Save the configuration file.

```
<config>
<in_path>384x288_Baseline3.0.mp4,640x360_Baseline3.0.mp4</in_path>
<bit_rates>550,1000</bit_rates>
<top_level_prefix>abr_set</top_level_prefix>
<out_type>hls</out_type>
```

```
<out_path>mbr_hls</out_path>
</config>
```

4. Run the Offline Packager, using the following command:

```
java -jar OfflinePackager.jar -conf_path config.xml
```

This generates the HLS output in the out_path/bit_rates directory. It creates a set level playlist (abr_set.m3u8) in the out_path folder. The variant play list created contains the information about the media files as follows:

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=550000
550/384x288_Baseline3.0.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1000000
1000/640x360_Baseline3.0.m3u8
```

Packaging multi-bit rate for HDS

1. To enable multi-bit rate packaging, encode a media file at different bit rates.
In this example, we have two files. They are encoded in 550 kbps and 1000 kbps respectively.
2. Place the input files in different directories. The in_path argument is relative to the command line location, so use an absolute path if necessary.
In the following example, the 550 kbps file is in the mbr_hds/500 directory and the 1000 kbps file is in the mbr_hds/1000 directory.
3. Create a configuration file to package the files by specifying the bit rates. Save the configuration file.

```
<config>
<in_path>384x288_Baseline3.0.mp4,640x360_Baseline3.0.mp4</in_path>
<bit_rates>550,1000</bit_rates>
<top_level_prefix>abr_set</top_level_prefix>
<out_type>hds</out_type>
<out_path>mbr_hds</out_path>
</config>
```

4. Run the Offline Packager, using the following command:

```
java -jar OfflinePackager.jar -conf_path config.xml
```

This generates the HDS output in the out_path/bit_rates directory. It creates a set level manifest file (abr_set.f4m) that contains information of the HDS files in the out_path tag specified.

```
<manifest xmlns="http://ns.adobe.com/f4m/2.0" version="2.0">
<media bitrate="550" href="550/384x288_Baseline3.0.f4m"> </media>
<media bitrate="1000" href="1000/640x360_Baseline3.0.f4m"> </media>
</manifest>
```

Manage various tracks in a stream

By default, the Offline Packager packages the first video and the audio tracks. However, you can package any audio or video track available in the source content or source media.

This can be useful in the following scenarios:

- Extracting audio from a file that contains both audio and video tracks.

For example, you can add audio-only stream as the lowest bitrate rendition in the HLS variant playlist.

- Packaging one video rendition by selecting a video track from video tracks of multiple bitrates or camera angles.
- Selecting audio for a specific language from multiple audio tracks.

If there are more than one track or stream in the source file, use the `track_ids` option to designate them for packaging. This parameter accepts the following values:

- For MP4 files: track IDs
- For TS files: stream PIDs

If the input file contains multiple audio or video, it packages the first video and the first audio PIDs, or tracks, or streams that are encountered. No error is displayed. The language of the selected audio track is parsed from the source file when available. To override the value in the source file, use the `lang` option and set the `lang` field accordingly.



Note: Specify at most one track ID of type audio and at most one track ID of type video.

For example, the following command packages the tracks with ID 1 and ID 2 of an MP4.

```
java -jar OfflinePackager.jar -in_path "D:\multiTrack.mp4" -out_type hds -out_path "C:\hds"
-track_ids=1,2
```

The following command packages third track containing French audio for a MPEG-DASH stream.

```
java -jar OfflinePackager.jar -in_path "D:\multiTrack.mp4" -out_type dash -out_path frenchAudio/
-frag_dur 4 -track_ids 3 -lang fr
```

Find the track information

You can use the `info` option to retrieve the information about all streams or tracks of an input file.

Use the following command to find the track information:

```
java -jar OfflinePackager.jar -in_path "D:\multiTrack.mp4" -info
```

Information similar to the following is returned:

```
Information about all the stream tracks in file D:\multiTrack.mp4
Number of tracks=2
Stream-Id=1, type=VIDEO, codec=H.264, lang=eng
Stream-Id=2, type=AUDIO, codec=AAC, lang=eng
```

PlaylistCreator tool

The PlaylistCreator tool is an additional tool to create top level manifest for packaged content. You can use the tool to create top level manifest for HDS, HLS, and MPEG-DASH.

The rationale for having set level manifest creation as a different step is that Offline Packager has a flat configuration structure (to enable input through the command-line or an XML file). For this reason, rendition-specific options cannot be specified while packaging. Each rendition must be packaged separately with its individual option. The set-level manifest creation must be performed later as a different step separate from packaging. The PlaylistCreator tool is a command line tool to enable you to develop scripts for the set-level manifest creation step.

To create Trick play streams, see [Generate a trick play stream](#).

The following configuration parameters are available for the PlaylistCreator tool:

| Parameter name | Description | Data type | Type |
|------------------|--|-----------|--|
| association | It is a list of semi-colon separated association, given in the format <media index>:a=<group id>,v=<group id>,s=<group id>. The media-index is 1-based index from the <code>-in_path</code> list and the group IDs are as given in the groups file. It is an HLS-only parameter and is used to associate group renditions to HLS stream files. The options a, v, and s indicate audio, video, and subtitles group types, respectively. A maximum of three association can be given for a media--one each for audio, video, and subtitles. However, providing association for all the three types is not mandatory. | String | Mandatory if <code>-group_files</code> is provided. |
| create_group | Option/flag to only create groups for the given set of playlist files. | NA | |
| create_playlist | Option/flag to create set level manifest/master playlist file | NA | |
| conf_path | Specify the path of the configuration file to be used for parameters. A relative path is assumed, unless you explicitly provides an absolute location of the configuration file. | String | Optional |
| in_path | Comma separated list of file-paths of input playlists. This should be the path of packaged contents. | String | Mandatory |
| out_path | Out path for playlist/group file. | String | Mandatory |
| out_type | Set the output type. Available types that are supported are HDS, HLS, and DASH. | String | Mandatory |
| top_level_prefix | Filename prefix of set level manifest or a master playlist. | String | Optional. Default "setLevel". |
| base_path | Base file path for list of files given in <code>in_path</code> . | String | Mandatory |
| id | Id for the group to be created to be used for HLS variant playlist. | String | Valid and mandatory for <code>-create_group</code> option. |
| default | HLS only parameter for <code>-create_group</code> option. List of media-index (1-based index in the <code>-in_path</code> list) to be marked as DEFAULT in the group. | String | Optional |
| auto | HLS only parameter for <code>-create_group</code> option. List of media-index (1-based index in the <code>-in_path</code> list) to be marked as AUTOSELECT in the group. | String | Optional |
| group_file_name | Name of the output group file for <code>-create_group</code> option. | String | Valid and mandatory for <code>-create_group</code> option. |
| alternate | HDS only parameter. Comma-separated list of media-index (1-based index in the <code>-in_path</code> list) to be marked as alternate. | String | Optional |
| group_files | HLS only parameter. Comma separated list of file-path of the group files for <code>-create_playlist</code> option. This path has to be either absolute or relative to the <code>base_path</code> . | String | Optional |
| validate | Validates the provided configuration file. If specified, only the configuration is validated. Creation of the playlist is omitted. | Boolean | Optional |

The following is a sample command to invoke the tool:

```
Command:
java -jar PlaylistCreator.jar -create_playlist -in_path media1.f4m,media2.f4m,media3.f4m
-out_type hds -out_path ~/Test/Output/hds -base_path ~/Test/Output/hds/media
```

The above command creates a file setLevel.f4m at ~/Test/Output/hds location:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns="http://ns.adobe.com/f4m/2.0" version="2.0">
  <media bitrate="500" href="media1.f4m"> </media>
  <media bitrate="1000" href="media2.f4m"> </media>
  <media bitrate="1500" href="media3.f4m"> </media>
</manifest>
```

Configuration file

Optionally, you can specify the configuration in a config.xml file. The parameter names to be used in the config.xml file are the same as the command line parameter names. For the command line flags(command line parameters which do not take any argument), to enable the flag in config file, just the tag should be provided.

```
java -jar PlaylistCreator.jar -conf_path path_of_conf_file
```

A sample configuration file for HDS (for the above example):

```
<config>
<create_playlist/>
<in_path>media1.f4m,media2.f4m,media3.f4m</in_path>
<out_type>hds</out_type>
<out_path>~/Test/Output/hds</out_path>
<base_path>~/Test/Output/hds/media</base_path>
</config>
```

You can specify both the config.xml file and the command line arguments. The command line arguments take precedence over those in the config file. Users can use this feature to put the common config for different output types in a single config file. For example, to create AAXS protected HLS output using same certificates, content protection config can be put in a single xml and then that config xml can be used for HLS packaging and set level creation.

Sample common content protection config.xml:

```
<config>
<drm/>
<lic_svr_cer>/Certificates/phds_license_server.der</lic_svr_cer>
<lic_svr_pfx>/Certificates/phds_license_server.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>FbsdseTCIZM=</lic_svr_pfx_pwd>
<pkgr_pfx>/Certificates/phds_production_packager.pfx</pkgr_pfx>
<pkgr_pfx_pwd>FbPMscTCIZM=</pkgr_pfx_pwd>
<transport_cer>/Certificates/phds_production_transport.der</transport_cer>
<common_key>/Certificates/commonkey.bin </common_key>
<policy_file>/Certificates/phds_24hr_policy.pol</policy_file>
<rec_cer>/Certificates/sd</rec_cer>
<content_id>primetime_demo_content</content_id>
</config>
```

You can now create an HLS set level playlist using the following command:

```
Command:
java -jar PlaylistCreator.jar -create_playlist -base_path ~/Test/Output -in_path
~/Test/Output/hls/30Sec/30Sec.m3u8 -out_path ~/Test/Output -out_type hls -top_level_prefix
hlsTest -conf_path ~/Test/config.xml
Output: hlsTest.m3u8

#EXTM3U
#EXT-X-VERSION:4
#EXT-X-FAXS-CM:URI="hls/30Sec/30Sec.drm"
```

```
#EXT-X-STREAM-INF:BANDWIDTH=2095422,PROGRAM-ID=1,CODECS="mp4a,avc1"
hls/30Sec/30Sec.m3u8
```

For the command line flags, the precedence rule is different in the sense that the flag is considered turned on if it's present in either the `config.xml` file or on the command line. It is considered off if and only if it is missing from both the config file and the command line.



Note: *PlaylistCreator tool depends on the `.properties` file to create top level manifest/playlist (for meta-data like language,label, bitrate etc.). The tool returns an error if no properties file is found at the `in_path`. Users are supposed to run `OfflinePackager` with `-properties_only` option to create a properties file before using the tool. If a user wants to customize any of the metadata values, the same needs to be manually edited in the respective properties file before running the tool.*

Specify alternate media for HDS

To specify alternate media, `-alternate` option can be used. The value is a comma separated list of media index from the `-in_path` value. Sample `config.xml` for packaging two MBR renditions with one alternate audio for HDS:

```
<config>
<create_playlist/>
<in_path>sample1.f4m,sample2.f4m,audio1/audioTrack.f4m</in_path>
<out_type>hds</out_type>
<out_path>~/Test/Output/hds</out_path>
<base_path>~/Test/Output/hds/altAudio</base_path>
<alternate>3</alternate>
</config>
```

For this configuration, the playlist creator will generate the following HDS set level manifest at the `out_path` with the name `setLevel.f4m` (default name as no `top_level_prefix` was provided):

Command:

```
java -jar PlaylistCreator.jar -conf_path ~/Test/config.xml
```

Output:

The set level manifest created is:

```
<?xml version="1.0" encoding="UTF-8"?><manifest xmlns="http://ns.adobe.com/f4m/2.0"
version="2.0">
<media bitrate="1039" href="sample1.f4m"> </media>
<media bitrate="1039" href="sample2.f4m"> </media>
<media alternate="true" bitrate="128" href="audio1/audioTrack.f4m" label="spanish" lang="spa"
type="audio"> </media>
</manifest>
```

Specify alternate media for HLS

Alternate media feature in HLS allows a provider to specify one of a set of variant playlists as an "override" (alternate) of the main presentation. The client will only play the alternate media (audio or video), and suppress any media of the same type from the main presentation, if present.

A new `EXT-X-MEDIA` tag has been defined for the variant playlist that identifies a media selection group. A set of `EXT-X-MEDIA` tags with the same `GROUP-ID` value forms a group of renditions. Each member of the group must represent an alternative rendition of the same content. In addition, two new attributes have been defined for the `EXT-X-STREAM-INF` tag. `AUDIO` specifies the audio media group and `VIDEO` specifies the video media group. These define the media options available while playing the stream. The `ASTREAM-INF` variant can indicate that it offers a choice of audio (or video) with an `AUDIO` (or `VIDEO`) attribute. This value is a group-id shared by every `MEDIA` tag that can be chosen.

For HLS, the Playlist creator tool lets you create groups using the `-create_group` option.

The following is a sample `config.xml` file, located at `~/Test/config.xml`. This sample is used to create an audio group with two alternates:

```
<config>
<create_group/>
<in_path>aud1/input_2v_2a_2cc.m3u8,/aud2/input_2v_2a_2cc.m3u8</in_path>
<out_type>hls</out_type>
<out_path>~/Test/Output/hls</out_path>
<base_path>~/Test/Output/hls/altAudio</base_path>
<id>group1</id>
<default>1</default>
<auto>1</auto>
<group_file_name>groups.xml</group_file_name>
</config>
```

The following is a sample `groups.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<groups>
  <group id="group1">
    <media autoselect="YES" codec="mp4a" default="YES" lang="eng" name="input_aud1"
type="AUDIO" uri="/aud1/input_2v_2a_2cc.m3u8"/>
    <media autoselect="NO" codec="mp4a" default="NO" lang="spa" name="input_aud2"
type="AUDIO" uri="/aud2/input_2v_2a_2cc.m3u8"/>
  </group>
</groups>
```

Creating variant playlist

You can create a variant playlist with an optional group file (using the parameter `-group_files`). Use the parameter association with `-create_playlist` option if `-group_files` is provided.



Note: Shell uses a semicolon to separate the commands. So quote the list of associations provided on the command line, to avoid the shell interpreting it as a semicolon.

A sample configuration file is shown below.

```
<config>
<create_playlist/>
<in_path>sample1.m3u8,sample2.m3u8,sample3.m3u8</in_path>
<out_type>hls</out_type>
<out_path>~/Test/Output/hls</out_path>
<base_path>~/Test/Output/hls/test</base_path>
<top_level_prefix>test</top_level_prefix>
<group_files>groups.xml</group_files>
<association>1:a=group1;2:a=group1;3:a=group1</association>
</config>
```

A sample command to run the PlaylistCreator is:

```
java -jar PlaylistCreator.jar -conf_path ~/Test/config.xml
```

The variant playlist created using the above sample configuration file is.

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="group1",NAME="input_aud1",AUTOSELECT=YES,DEFAULT=YES,LANGUAGE="eng",URI="/aud1/input_2v_2a_2cc.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="group1",NAME="input_aud2",AUTOSELECT=NO,DEFAULT=NO,LANGUAGE="spa",URI="/aud2/input_2v_2a_2cc.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=1136773,PROGRAM-ID=1,AUDIO="group1"
sample1.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=137227,PROGRAM-ID=1,AUDIO="group1"
sample2.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=137528,PROGRAM-ID=1,AUDIO="group1"
sample3.m3u8
```

You can also use WebVTT files for subtitles. Use the option 's' to indicate subtitles. For example, if a group ID is 'subs' and there are two main media renditions, then the association parameter is `-association 1:a=group1,s=subs;2:a=group1,s=subs` in the `-create_playlist` command.

MPEG-DASH support

The Playlist Creator tool supports creation of master MPD by merging MPDs, containing audio/video/trick play renditions and closed captions.

Playlistcreator generates master MPD by parsing all the rendition MPD files. Unlike for HDS and HLS, it doesn't process .properties files. All video renditions are automatically added by PlaylistCreator in one AdaptationSet in the final MPD created by it. Playlistcreator assumes that all video renditions, if encrypted, had same DRM configuration during packaging. There is no action required from user for this. It contains different AdaptationSet, one for each language, for audio streams, and for Closed Captions.

All video Representations are added to one AdaptationSet. Multiple camera angles are not supported. Similarly all trickplay renditions are added in one AdaptationSet.

Only one audio Representation for a given language is added to master MPD, even if there are multiple such Representations in the source MPDs. The ID of each AdaptationSet and all contained Representations are made unique in the final MPD. The BaseURL element of each Representation is modified based on the location of the containing stream-level MPD, relative to the folder specified using the `base_path` parameter.

A sample command is that generates setLevel.mpd file in the folder C:\PackagedMedia is:

```
java -jar PlaylistCreator.jar -out_type dash -base_path C:\PackagedMedia -out_path
C:\PackagedMedia -in_path
trickplay1.mpd,trickplay2.mpd,audioFr.mpd,captions.mpd,loBitrate.mpd,midBitrate.mpd,highBitrate.mpd
-create_playlist
```

For the sample given above, the following table shows the structure of the master MPD based on the contents of the stream level MPDs.

Table 2: List of contained Representations in the stream-level MPDs based on the above sample

| Stream level MPD | Contained Representation |
|------------------|--------------------------|
| trickplay1.mpd | R_trickplay1 |
| trickplay2.mpd | R_trickplay2 |
| audioFr.mpd | R_audio1(fr) |
| captions.mpd | R_cc1(en) |
| loBitrate.mpd | R_audio2(en), R_video1 |
| midBitrate.mpd | R_audio3(en), R_video2 |
| highBitrate.mpd | R_audio4(en), R_video3 |

The structure of the Master MPD based on the above sample is:

```
AdaptationSet1 { R_trickplay1, R_trickplay2 }
AdaptationSet2(fr) { R_audio1 }
AdaptationSet3(en) { R_ccl }
AdaptationSet4(en) { R_audio2 }
AdaptationSet5 { R_video1, R_video2, R_video3 }
```

Playlistcreator can be used to provide alternative BaseURLs to indicate availability of identical packaged content at multiple locations as a failover support. Multiple BaseURLs are specified as value of `add_base_urls` parameter, a list of URLs separated by double semicolons(;). The provided URLs must contain a scheme like `http://` or `https://` and any relative URLs without scheme are not added.

A sample command to create the playlist is:

```
java -jar PlaylistCreator.jar -out_type dash -create_playlist ..... -add_base_urls
'http://server1.com/basePath/;http://www.server2.com/backup/'
```

These URLs are added at MPD level in multiple BaseURL elements as shown below:

```
<MPD xmlns="urn:mpeg:DASH:schema:MPD:2011" xmlns:scte35="urn:scte:scte35:2013"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" mediaPresentationDuration="PT298S"
minBufferTime="PT4S" profiles="urn:mpeg:dash:profile:isoff-live:2011" type="static"
xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 DASH-MPD.xsd">
<BaseURL>http://server1.com/basePath/</BaseURL>
<BaseURL>http://www.server2.com/backup/</BaseURL>
<Period id="1" start="PT0S">
....
</Period>
</MPD>
```



Note: Shell uses ';' to separate commands, so when the list of base URLs is provided on the command line it must be quoted to avoid semicolon interpretation by Shell.

Content protection

Offline Packager supports content protection using Adobe Access, FairPlay, PlayReady, and Widevine. It supports AES-128 and Sample-AES encryption techniques to protect without DRM. Refer to the table below for the supported methods.

Table 3: Primetime content protection technology support

| DRM applicability | DASH | HLS | HDS |
|-------------------|---------------------------------------|------------------------------------|--------------------|
| With DRM | Adobe Access, Playready, and Widevine | Adobe Access and FairPlay | Adobe Access |
| Without DRM | Common encryption | Sample-AES and AES-128 encryptions | AES-128 encryption |

Protect content using AAXS DRM

1. Create a configuration file.
 - a) Enable DRM by providing the `drm` tag.
 - b) Provide the following options:
 - `content_id` - a content identifier you select. It's used to generate the content encryption key.

- `lic_svr_url` - a URL of the Flash Access for Protected Streaming Adobe Access License Server. It grants the user license.
- `lic_svr_cer` - an encoded Adobe Access License Server certificate. It is obtained from Adobe as a result of licensing and never changes.
- `lic_svr_pfx` - Adobe Access License Server credentials path.
- `lic_svr_pfx_pwd` - Adobe Access License Server credentials password.
- `pkgr_pfx` - The PFX file containing the packager's protection credentials.
- `pkgr_pfx_pwd` - The password string used to secure the packager credentials.
- `transport_cer` - is an encoded transport certificate (.der). It is obtained from Adobe through licensing and never changes.
- `common_key_file` - is a unique 128-bit key that's used with the content-id to create the encryption key. It is a random binary file that contains at least 16 bytes and can be created manually.

The bytes in the `common_key_file` should have sufficient entropy or randomness to ensure a unique encryption key.

- `policy_file` - policy (.pol). The policy file can be created using the java API or a utility that comes with Flash Access (AdobePolicyManager.jar).



Note: Use the same parameters for all content in the MBR set.

Sample configuration file:

```
<config>
<out_path>contentProtection/Embed_Valid_hls</out_path>
<out_type>hls</out_type>
<drm>true</drm>
<drm_sys>AAXS</drm_sys>
<content_id>test</content_id>
<key_url>https://key_url/</key_url>
<lic_svr_url>license_server_url</lic_svr_url>
<lic_svr_cer>license_server_certificate.der</lic_svr_cer>
<lic_svr_pfx>license_server_credentials.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>kY2IUPnQuG0=</lic_svr_pfx_pwd>
<pkgr_pfx>packager_credential_path.pfx</pkgr_pfx>
<pkgr_pfx_pwd>kY2IUPnQuG0=</pkgr_pfx_pwd>
<transport_cer>transport_certificate_path.der</transport_cer>
<common_key_file>common_key_file_path.bin</common_key_file>
<policy_file>policy_file.pol</policy_file>
</config>
```

2. Run Offline Packager with the configuration file.

External CEK support to package HLS content with AAXS DRM

External CEK support allows for packaging of content with AAXS DRM and to create metadata that does not contain the CEK (protected with an Adobe Primetime DRM license server's public certificate). The tool embeds a CEK ID into the content's metadata. During license acquisition, license server observes a flag in the metadata to identify that the content is protected using an External CEK. The license server extracts the CEK ID from the metadata and then queries a secure repository/CKMS to retrieve the appropriate CEK.

See the [External CEK overview](#) to get an overview of the External CEK feature. To package the content using External CEK, use `aaxs_external_cek` flag. Also, `key_id` parameter becomes applicable for AAXS with External CEK. You can specify the parameters on the command line or in the configuration XML file.

An example of the command line is below:

```
java -jar OfflinePackager.jar -key_id somekeyID -aaxs_external_cek -conf_path
aaxs_drm_settings_config_file_path
```

A sample configuration file for HLS is:

```
<config>
<out_type>hls</out_type>
<in_path>input_file_path</in_path>
<out_path>output_file_path</out_path>
<drm/>
<drm_sys>AAXS</drm_sys>
<content_id>test</content_id>
<lic_svr_url>http://example.com:8080</lic_svr_url>
<lic_svr_cer>~/Primetime-license-PRO-20151027.der</lic_svr_cer>
<lic_svr_pfx>~/Primetime-license-PRO-20151027.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>ZVDnkTU0ulc=</lic_svr_pfx_pwd>
<pkgr_pfx>~/Primetime-packager-PRO-20151027.pfx</pkgr_pfx>
<pkgr_pfx_pwd>nVcOW3U6TAS=</pkgr_pfx_pwd>
<transport_cer>~/Primetime-transport-PRO-20151027.der</transport_cer>
<common_key_file>~/common-key.bin</common_key_file>
<policy_file>~/extCekLicChain.pol</policy_file>
<rec_cer>~/sd</rec_cer>
<key_file_path>~/key.bin</key_file_path>
<key_id>keyId</key_id>
<aaxs_external_cek/>
</config>
```

Protect content using Widevine DRM for DASH

Primetime supports Widevine DRM for DASH. Widevine DRM provides the capability to license, securely distribute, and protect playback of content on multiple platforms.

From packaging perspective, the key file contains a Base64 encoded version of CEK file. The CEK is a (random) 16 byte value and must be coded in Hex (Base16).

Use either of the following two command lines for offline packager:

- Provide the values for the provider, the content_id, and so on, to make a Widevine header.

```
java -jar OfflinePackager.jar -in_path sample.mp4 -out_type dash -out_path out_file_path
-drm -drm_sys WIDEVINE -key_file_path "creds/key.bin" -widevine_key_id somekeyID
-widevine_content_id contentID -widevine_provider drmProviderName
```

- Provide a Widevine header.

```
java -jar OfflinePackager.jar -in_path sample.mp4 -out_type dash -out_path out_file_path
-drm -drm_sys WIDEVINE -key_file_path "creds/key.bin" -widevine_key_id somekeyID
-widevine_header WidevineHeader
```

It is mandatory to provide key_id and key_file_path in both cases.

The sample configuration file corresponding to the former case is below:

```
<config>
<in_path>sample.mp4</in_path>
<out_type>dash</out_type>
<out_path>out_file_path</out_path>
<drm/>
<drm_sys>widevine</drm_sys>
<frag_dur>4</frag_dur>
<target_dur>6</target_dur>
<key_file_path>creds/widevine.bin</key_file_path>
<widevine_content_id>contentID</widevine_content_id>
<widevine_provider>drmProviderName</widevine_provider>
```

```
<widevine_key_id>somekeyID</widevine_key_id>
</config>
```

The sample configuration file corresponding to the latter case is below:

```
<config>
<in_path>sample.mp4</in_path>
<out_type>dash</out_type>
<out_path>out_file_path</out_path>
<drm/>
<drm_sys>widevine</drm_sys>
<frag_dur>4</frag_dur>
<target_dur>6</target_dur>
<key_file_path>creds/widevine.bin</key_file_path>
<widevine_key_id>somekeyID</widevine_key_id>
<widevine_header>WidevineHeader</widevine_header>
</config>
```

Protect content using Sample-AES encryption

Sample-AES encryption is implemented according to specifications at [MPEG-2 Stream Encryption Format for HTTP Live Streaming](#).

The command line for Sample-based AES usage is: `java -jar OfflinePackager.jar -in_path sample.mp4 -out_type hls -out_path out_file_path -drm -drm_sys none -hls_enc_type sample`

The generated m3u8 file has EXT-X-KEY attribute as follows, where key file is simply served by a file URI.

```
#EXT-X-KEY:METHOD=SAMPLE-AES,URI="encrypt_key_0.bin",IV=some_value
```

The Packager generates the key file if the key file path is not specified. A sample configuration file for reference is provided below:

```
<config>
<in_path>sample.mp4</in_path>
<out_type>hls</out_type>
<out_path>sample</out_path>
<drm/>
<drm_sys>none</drm_sys>
<hls_enc_type>sample</hls_enc_type>
<frag_dur>4</frag_dur>
<target_dur>6</target_dur>
<content_id>_default_</content_id>
</config>
```

Protect content using FairPlay DRM

Publishers can protect their content using Apple's FairPlay DRM.

The command to create the stream using FairPlay is: `java -jar OfflinePackager.jar -in_path sample.mp4 -out_type hls -out_path out_file_path -drm -drm_sys FAIRPLAY -key_file_path "creds/fairplay.bin" -key_url "user_provided_value"`

The generated m3u8 file has EXT-X-KEY attribute as follows:

```
#EXT-X-KEY:METHOD=SAMPLE-AES,URI="user_provided_value",
KEYFORMAT="com.apple.streamingkeydelivery",KEYFORMATVERSIONS="1"
```

It is mandatory to provide the key_url value, which is then copied as is in the m3u8 file. A sample configuration file for reference is provided below:

```
<config>
<in_path>mp4_file_path</in_path>
<out_type>hls</out_type>
```

```
<out_path>out_file_path</out_path>
<drm/>
<drm_sys>FAIRPLAY</drm_sys>
<frag_dur>4</frag_dur>
<target_dur>6</target_dur>
<key_file_path>creds/fairplay.bin</key_file_path>
<iv_file_path>creds/iv.bin</iv_file_path>
<key_url>user_provided_value</key_url>
<content_id>_default_</content_id>
</config>
```

Protect content using multi-DRM for HLS (AAXS and FairPlay)

Offline Packager provides multi-DRM support in a single manifest where the content is encrypted only once. In accordance with [these HLS specifications](#), EXT-X-KEY tag can be used to achieve multi-DRM.



Note: The multi-DRM content packaged by Offline Packager is not supported by the TVSDK client. The TVSDK client does not recognize the `KEYFORMAT` attribute used in such packaging.

Offline Packager adds multiple key tags with different key formats in the m3u8 file to achieve multi-DRM.

The generated m3u8 file has multiple EXT-X-KEY tags. Key rotation is not supported with this method.

```
#EXT-X-FAXS-CM:MIIr0gYJKoZIhvcNAQcCoIIrwcCCK78CAQE...=
#EXT-X-KEY:METHOD=SAMPLE-AES,URI="user_provided_value",
KEYFORMAT="com.apple.streamingkeydelivery",KEYFORMATVERSIONS="1"
#EXT-X-KEY:METHOD=SAMPLE-AES,URI="user_provided_value",
KEYFORMAT="com.adobe.access",KEYFORMATVERSIONS="1"
```

A sample configuration file for reference is provided below:

```
<config>
<in_path>video.mp4</in_path>
<out_type>hls</out_type>
<out_path>multidrm</out_path>
<drm/>
<inline_drm/>
<drm_sys>aaxs,fairplay</drm_sys>
<hls_enc_type>sample</hls_enc_type>
<frag_dur>4</frag_dur>
<target_dur>6</target_dur>
<key_url>user_provided_value</key_url>
<lic_svr_url>http://example.com:8080/flashaccessserver/axs_prod</lic_svr_url>
<lic_svr_cer>creds/static/Primetime-license-PRO-20151027.der</lic_svr_cer>
<lic_svr_pfx>creds/static/Primetime-license-PRO-20151027.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>ZVDnkTU0ulc=</lic_svr_pfx_pwd>
<pkgr_pfx>creds/static/Primetime-packager-PRO-20151027.pfx</pkgr_pfx>
<pkgr_pfx_pwd>nVcOW3U6TAs=</pkgr_pfx_pwd>
<transport_cer>creds/static/Primetime-transport-PRO-20151027.der</transport_cer>
<common_key_file>creds/common-key.bin</common_key_file>
<policy_file>creds/static/phds_policy.pol</policy_file>
<key_file_path>key.bin</key_file_path>
<iv_file_path>iv.bin</iv_file_path>
<content_id>fnewflknslkfsd</content_id>
</config>
```

Protect DASH content using AAXS and Playready DRM

The Offline Packager supports [Playready DRM](#) for DASH packaging. Common Encryption Scheme, as defined in [ISO/IEC 23001-7:2015 CENC specifications](#), specifies AES-CTR mode encryption, sub-sample mapping and key identification for DASH that is common to all DRM systems.

You can specify a custom Content Encryption Key (CEK) for DASH, using the parameter `key_file_path`. This is applicable for AAXS and PlayReady DRM systems, though with minor differences, such as the size of Initialization Vector (IV) is 16 for AAXS and 8 for PlayReady. For MPEG DASH packaging, the parameter `drm_sys` can be set to `aaxs` or `playready_buydrm` as required. The Offline Packager implements PlayReady DRM and CENC encryption as documented by Microsoft in [Content protection for MPEG DASH](#) and [Playready Header Object \(PRO\)](#).

BuyDRM PlayReady support

The Offline Packager supports the Playready encryption integrated with BuyDRM workflow. The following is a quick overview of the BuyDRM workflow for securing key generation, encryption and license acquisition:

- The Offline Packager generates a SOAP request containing KID, UserKey, Content-id, and posts to KeyOS service. The packager extracts Content Key and Playready Header Object (PRO) from the response.
- The content is DASH packaged and encrypted using the above Content Key. DRM specific fields are generated using the Playready Header (obtained in the response). Offline Packager replaces the license server URL (LA_URL element) by the proxy server URL.
- Before playing back the protected media, a player based on Primestime TVSDK sends the license acquisition request to the proxy server.
- The proxy server augments the request with an Authentication XML signed using an RSA keypair that is whitelisted by BuyDRM. Authentication XML is a short-lived security token. The augmented request is forwarded to BuyDRM/KeyOS license server.
- The license server verifies all the parameters in the augmented request and grants license appropriately. The license is returned back to the player by the proxy server.

Offline Packager configuration

To use BuyDRM protection, set the value of `drm_sys` parameter to `playready_buydrm`. The parameter `buydrm_proxy_url` is optional and is required only when the proxy server is deployed. Customer specific UserKey, assigned by BuyDRM, is provided using the option `buydrm_userkey`. The option `buydrm_mediaid`, that denotes the media identification, is set to value of `in_path` parameter when omitted. Content ID and Key ID if provided, must be in the GUID format where 16 bytes are listed as Hexadecimal digits and the fields of 4, 2, 2, 2, and 6 bytes are separated by dash. An example of a GUID is 3F2504E0-4F89-41D3-9A0C-0305E82C3301.

To package multiple video bitrate renditions protected using the same DRM, ensure that all BuyDRM specific parameters, except `buydrm_mediaid`, are identical for each invocation of Offline Packager. To package multiple renditions of the same content with same key, provide the Key ID. If the Key ID is not provided, a new Key ID is generated for each rendition and a separate Content Key is requested from BuyDRM for each rendition.

Sample configuration file with `playready_buydrm` protection

Based on the above suggested configurations, a sample file is provided below:

```
<config>
<in_path>source.mp4</in_path>
<out_type>dash</out_type>
<out_path>buydrm_encrypted</out_path>
<drm/>
<drm_sys>playready_buydrm</drm_sys>
<frag_dur>4</frag_dur>
<buydrm_contentid>961cc844-0f34-2f41-b37c-74f3b0f192b1</buydrm_contentid>
<buydrm_userkey>18a33f53-dc90-42e0-bc22-2c2f450c4971</buydrm_userkey>
<buydrm_keyid>2cc84864-0f38-8f40-b37d-f3b0f19274b1</buydrm_keyid>
<buydrm_mediaid>test_file</buydrm_mediaid>
<buydrm_proxy_url>http://sample.proxyserver.com</buydrm_proxy_url>
</config>
```

Protect content using multi-DRM for DASH (AAXS, Playready, and Widevine)

Primetime support multi-DRM functionality for DASH streams. It means that in a single manifest (MPD) file, with one-time encryption, different DRMs can be mentioned. The supported DRM systems for DASH are Playready, Widevine, and AAXS. Mention the applicable values in the `drm_sys` parameter as a comma separated list. The applicable values are `widevine`, `aaxs`, `playready`, or any combination of these values.

A sample configuration file that uses Intertrust as a provider for Playready and Widevine DRM is below.

```
<config>
<in_path>sample.mp4</in_path>
<out_type>dash</out_type>
<out_path>out_file_path</out_path>
<drm/>
<drm_sys>widevine,aaxs,playready</drm_sys>
<frag_dur>4</frag_dur>
<target_dur>6</target_dur>
<key_file_path>playready.bin</key_file_path>
<encrypt_audio>false</encrypt_audio>
<widevine_content_id>contentID</widevine_content_id>
<widevine_provider>intertrust</widevine_provider>
<!-- widevine_key_id and playready_keyid are deprecated. Use single config key_id instead. -->
<!-- <widevine_key_id>widevineKeyID</widevine_key_id>
<playready_keyid>playreadyKeyID</playready_keyid> -->
<key_id>8aef14288304aef74ea2c3b5200e3b15f</key_id>
<playready_LA_URL>playreadyLAURL</playready_LA_URL>

<lic_svr_url>http://example.com:8096</lic_svr_url>
<lic_svr_cer>creds/static/AAXS-PROD_CDM-PRO-LIC-20141016.der</lic_svr_cer>
<lic_svr_pfx>creds/static/AAXS-PROD_CDM-PRO-LIC-20141016.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>certpassword</lic_svr_pfx_pwd>
<pkgr_pfx>creds/static/AAXS-PROD_CDM-PRO-PKGR-20141016.pfx</pkgr_pfx>
<pkgr_pfx_pwd>certpassword</pkgr_pfx_pwd>
<transport_cer>creds/static/AAXS-PROD_CDM-PRO-TSPT-20141016.der</transport_cer>
<common_key_file>creds/common-key.bin</common_key_file>
<policy_file>creds/static/common.pol</policy_file>
<rec_cer>creds/sd</rec_cer>
<content_id>dash_vidoe_drm</content_id>
</config>
```

Note the following best practices and cautions when packaging to achieve multi-DRM with DASH:

- For multi-DRM with different key providers to work, use the same key ID and content key with all the key providers. For example, the Widevine content key and key ID must be the same as the PlayReady content key and key ID when using Widevine and Playread together. Note that Playready DRM accepts KID in GUID format as well, whereas Widevine accepts a Hex representation of GUID format in Big-endian format. For more details, see <https://msdn.microsoft.com/en-us/library/dn468966.aspx>.
- Playready key ID (GUID format) and Widevine key ID (UUID format) are supported as separate configuration params. Use any of these two IDs for multi-DRM case. The final generated content corresponds to a single KID and Key.
- Playready players have limitation of supporting 8 byte IV only. If Playready is specified as one of the DRM systems, 8 byte IV is used for Widevine and AAXS DRMs too.

DRM/FAXS impact

Primetime Offline Packager 1.2. or higher, does not create an expired leaf license for chained license policy. Leaf license properties, such as validity, start date, and end date are determined from the policy. To create an expired leaf license to enforce the license validity from root license, specify the end data in the policy.

Protect HLS content using AES-128 encryption

Advanced Encryption Standard (AES) is a specification for encrypting electronic data. Offline Packager supports vanilla AES-128 encryption for HLS streaming.

You don't need to use Adobe Access Server implementation for content protection. This encryption is supported while packaging an existing HLS stream or MP4. Vanilla HLS encryption can be done using the user-generated or packager-generated key types. For both the key types, the Initialization Vector (IV) is generated by Offline Packager.



Note: AES mode is only available for HLS output.

Encrypt with a user-generated key

1. Create a configuration file.

- a) Add the `drm` tag to enable content protection.
- b) Provide the value of the `drm_sys` tag as `none`.

By default the `drm_sys` tag's value is `AAXS`. By providing the value as `none`, you can provide your own key or make the system to generate one.

- c) Provide the key location in the `key_file_path` tag. If you don't provide this value, packager automatically generates a key.
- d) Provide the `key_url` value if you want to append the URL in m3u8 file generated.

Sample file:

```
<config>
<in_path>384x288_Baseline3.0.mp4</in_path>
<out_type>hls</out_type>
<out_path>384x288_Baseline3.0_mp4_hls_user_gen</out_path>
<drm/>
<drm_sys>none</drm_sys>
<frag_dur>8</frag_dur>
<target_dur>12</target_dur>
<key_file_path>secureLocation\myKey.base64</key_file_path>
<key_url></key_url>
</config>
```

2. Run the command to package the content.

Encrypt with a packager-generated key

In this scenario, the packager randomly generates the key for encryption.

1. Create a configuration file.

- a) Add the `drm` tag to enable content protection.
- b) Provide the value of the `drm_sys` tag as `none`.

By default the `drm_sys` tag's value is `AAXS`. By providing the value as `none`, you can provide your own key or make the system to generate one.

- c) (Optional) Provide the `key_out_path` tag to save the packager-generated keys.
If not provided, Offline Packager copies the generated key to the output directory specified.

- d) (Optional) Provide the `key_rot` tag to enable key rotation.

By default, the key rotation duration is 900 seconds. You can specify a different value using the `key_rot_dur` parameter.

- e) (Optional) Provide the `key_file_name` tag to prefix the key file.

By default, the naming convention of the generated key is encrypt_key_<key-index-number_for_key_rotation>.bin. If key rotation is enabled, an index number that increments each time the key is rotated is added with the file name.

f) (Optional) Add the key_base_url tag to prepend to the generated key name.

A sample configuration file:

```
<config>
<in_path>40x480_Main3.1.mp4</in_path>
<out_type>hls</out_type>
<out_path>test</out_path>
<drm/>
<drm_sys>none</drm_sys>
<frag_dur>8</frag_dur>
<target_dur>12</target_dur>
<key_out_path>/secure/location</key_out_path>
<key_rot/>
<key_file_name>enc</key_file_name>
<key_base_url></key_base_url>
</config>
```

2. Run the command to package the content.

Enable PHDS and PHLS

For Protected HDS (PHDS) and Protected HLS (PHLS) content, a license server is not required, because the license is embedded in the DRM meta data.

Specify a Policy file for which the LicenseBindingType is set for embedded license.

The Offline packager provides default packaging certificates in the install_dir/creds/static directory. You can use these certificate files for PHDS and PHLS. You can also specify a recipient certificate directory path. By default, the recipient certificates are packaged with Offline Packager in the install_dir/cred/sd folder. If you don't specify any recipient certification path, certificates are obtained from the default path.

A sample configuration file for PHLS is provided below:

```
<config>
<out_path>contentProtection/Embed_Valid_hls</out_path>
<out_type>hls</out_type>
<drm>true</drm>
<drm_sys>AAXS</drm_sys>
<content_id>test</content_id>
<key_url>https://key_url/</key_url>
<lic_svr_url>http://license_server_url/</lic_svr_url>
<lic_svr_cer>creds/static/phds_license_server.der</lic_svr_cer>
<lic_svr_pfx>creds/static/phds_license_server.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>kY2IUPnQuG0=</lic_svr_pfx_pwd>
<pkgr_pfx>creds/static/phds_production_packager.pfx</pkgr_pfx>
<pkgr_pfx_pwd>kY2IUPnQuG0=</pkgr_pfx_pwd>
<transport_cer>creds/static/phds_production_transport.der</transport_cer>
<common_key_file>creds/common-key.bin</common_key_file>
<policy_file>creds/static/phds_policy.pol</policy_file>
<rec_cer>creds/sd</rec_cer>
</config>
```

A sample configuration file for PHDS is provided below:

```
<config>
<out_path>contentProtection/Embed_Valid_hds</out_path>
<out_type>hds</out_type>
<drm>true</drm>
<drm_sys>AAXS</drm_sys>
<content_id>test</content_id>
```

```
<lic_svr_url>http://license_server_url</lic_svr_url>
<lic_svr_cer>creds/static/phds_license_server.der</lic_svr_cer>
<lic_svr_pfx>creds/static/phds_license_server.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>kY2IUPnQuG0=</lic_svr_pfx_pwd>
<pkgr_pfx>creds/static/phds_production_packager.pfx</pkgr_pfx>
<pkgr_pfx_pwd>kY2IUPnQuG0=</pkgr_pfx_pwd>
<transport_cer>creds/static/phds_production_transport.der</transport_cer>
<common_key_file>creds/common-key.bin</common_key_file>
<policy_file>creds/static/phds_policy.pol</policy_file>
<rec_cer>creds/sd</rec_cer>
</config>
```



Note: You can use the same policy files for both HLS and HDS.

ms_unencrypted support for HLS and HDS

You can use the configuration parameter `ms_unencrypted` in Offline Packager to specify the number of milliseconds at the beginning of the content that remains unencrypted.

You can use the `ms_unencrypted` option to improve stream start times because it allows the client to acquire the license asynchronously when the playback begins. Offline Packager 1.3 onwards, `ms_unencrypted` support is provided for both, HLS and HDS outputs. For HLS output, this configuration option works for both AAXS protected content and vanilla AES encrypted content. Because there is no partial encryption support for HLS, Adobe recommends that you provide integral multiples of the fragment duration as the value for the `ms_unencrypted` parameter. However, if the value `ms_unencrypted` does not align with the fragment boundaries, the fragments are not split. The amount of the content left unencrypted is an integral number of HLS segments, whose sum total of duration is lower bound by `ms_unencrypted` configuration value. For example, the value of the `ms_unencrypted` is specified as 12000 (12 seconds).

The following is a sample m3u8:

```
#EXTM3U#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:7
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-MEDIA-TIME:0.0#EXTINF:6.006,artbeats.0.ts
#EXTINF:3.003,artbeats.6006.ts
#EXTINF:3.003,artbeats.9009.ts
#EXTINF:6.006,artbeats.12012.ts
#EXTINF:3.003,artbeats.18018.ts
#EXTINF:3.003,artbeats.21021.ts
#EXT-X-ENDLIST
```

In this case, the number of fragments left unencrypted will be 3 and the sum total of their duration is:

```
6006 + 3003 + 6006 = 15015 which is lower bound by 12000.
```

If two fragments were left unencrypted, the sum of their duration would have been:

```
6006 + 3003 = 9009 which is not bound below by 12000.
```

The following is the sample contents of the m3u8 with the value of `ms_unencrypted` as 12000:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:7
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-FAXS-CM:URI="artbeats.drm"
#EXT-X-MEDIA-TIME:0.0
#EXTINF:6.006,artbeats.0.ts
#EXTINF:3.003,artbeats.6006.ts
#EXTINF:3.003,artbeats.9009.ts
#EXT-X-KEY:METHOD=AES-128,URI="faxes://faxes.adobe.com",IV=0x697c5ca69d276c71e249b627a66ad979
#EXT-X-MEDIA-TIME:12.012
```

```
#EXTINF:6.006,artbeats.12012.ts
#EXTINF:3.003,artbeats.18018.ts
#EXTINF:3.003,artbeats.21021.ts
#EXT-X-ENDLIST
```

Custom IV support

Offline Packager supports AES-128 encryption (with DRM) and Sample-AES for HLS output. By default, Offline Packager generates an Initialization Vector (IV). However, you can provide a custom IV. You can also output the automatically generated IV to an output file. For command line options, see [Configuration parameters](#).

Whitelist an application for Adobe Access

App whitelisting ensures that the protected content can be played only in the specified apps and SWFs.

To enable app whitelisting, specify the path to a folder containing the app.whitelist file in the whitelist parameter for PHLS. For PHDS, specify the path to the folder containing the SWF files.



Note: For SWF verification and app whitelisting to work simultaneously, place the app.whitelist file and SWF files in the same directory. For app whitelisting, keep the publisher ID in the same directory.

Enable app whitelisting for HLS

App whitelisting ensures that the protected content can be played only in the specified apps and SWFs.

Create an app.whitelist file that contains publisher ID, app ID, and minimum and maximum version of the app.

A sample app.whitelist file:

```
pub1_cert.cer, FJ8SDN490SJSD9HA83JA073JD03KSA0, 1.0, 4.0 pub2_cert.cer,
NJFS84NFW0942NFWJFW904FW904THFW
pub3_cert.cer
```

In the example provided above, you can playback the content only on:

- Versions 1.0 to 4.0 of the app with ID FJ8SDN490SJSD9HA83JA073JD03KSA0 signed by pub1_cert.cer.
- All versions of app with ID NJFS84NFW0942NFWJFW904FW904THFW signed by pub2_cert.cer.
- All apps signed by pub3_cert.cer.

A sample configuration file:

```
<config>
<out_path>contentProtection/Embed_Valid_hls</out_path>
<out_type>hls</out_type>
<drm>true</drm>
<drm_sys>AAXS</drm_sys>
<content_id>test</content_id>
<key_url>https://key_url/</key_url>
<lic_svr_url>http://license_server_url</lic_svr_url>
<lic_svr_cer>creds/static/phds_license_server.der</lic_svr_cer>
<lic_svr_pfx>creds/static/phds_license_server.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>kY2IUPnQuG0=</lic_svr_pfx_pwd>
<pkgr_pfx>creds/static/phds_production_packager.pfx</pkgr_pfx>
<pkgr_pfx_pwd>kY2IUPnQuG0=</pkgr_pfx_pwd>
<transport_cer>creds/static/phds_production_transport.der</transport_cer>
<common_key_file>creds/common-key.bin</common_key_file>
<policy_file>creds/static/phds_policy.pol</policy_file>
<rec_cer>creds/sd</rec_cer>
```

```
<Whitelist>C:/whitelistFolderForHLS</Whitelist>
</config>
```

Enable SWF verification for HDS

For HDS files, the content will playback only on the player that is used to package the SWF.

Specify the WhitelistFolder folder that contains the whitelisted SWFs.

A sample configuration file:

```
<config>
<out_path>contentProtection/Embed_Valid_hds</out_path>
<out_type>hds</out_type>
<drm>true</drm>
<drm_sys>AAXS</drm_sys>
<content_id>test</content_id>
<key_url>https://key_url/</key_url>
<lic_svr_url>http://license_server_url</lic_svr_url>
<lic_svr_cer>creds/static/phds_license_server.der</lic_svr_cer>
<lic_svr_pfx>creds/static/phds_license_server.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>kY2IUPnQuG0=</lic_svr_pfx_pwd>
<pkgr_pfx>creds/static/phds_production_packager.pfx</pkgr_pfx>
<pkgr_pfx_pwd>kY2IUPnQuG0=</pkgr_pfx_pwd>
<transport_cer>creds/static/phds_production_transport.der</transport_cer>
<common_key_file>creds/common-key.bin</common_key_file>
<policy_file>creds/static/phds_policy.pol</policy_file>
<rec_cer>creds/sd</rec_cer>
<Whitelist>C:/whitelistFolderForHDS</Whitelist>
</config>
```

Refresh the license for HDS and HLS

In Protected HTTP Dynamic Streaming (PHDS) and Protected HTTP Live Streaming (PHLS), the license is generated while packaging and is embedded with the DRM meta data of the content. The PHDS and PHLS policies allow license validity for a specified duration of time.

To extend the license duration using the Offline Packager, use the `drm_refresh` tag.

Provide the same information that you provided while generating the license. A sample configuration file:

```
<config>
<in_path>file.mp4</in_path>
<out_type>hds</out_type>
<out_path>/hds/primetime_demo_content</out_path>
<drm/>
<drm_sys>AAXS</drm_sys>
<drm_refresh/>
<lic_svr_cer>/Certificates/phds_license_server.der</lic_svr_cer>
<lic_svr_pfx>/Certificates/phds_license_server.pfx</lic_svr_pfx>
<lic_svr_pfx_pwd>FbsdseTCIZM=</lic_svr_pfx_pwd>
<pkgr_pfx>/Certificates/phds_production_packager.pfx</pkgr_pfx>
<pkgr_pfx_pwd>FbPMscTCIZM=</pkgr_pfx_pwd>
<transport_cer>/Certificates/phds_production_transport.der</transport_cer>
<common_key>/Certificates/commonkey.bin </common_key>
<policy_file>/Certificates/phds_24hr_policy.pol</policy_file>
<rec_cer>/Certificates/sd</rec_cer>
<content_id>primetime_demo_content</content_id> </config>
```



Note: To refresh the PHDS license every 24 hours, you can write a script to run Offline Packager based on the configuration provided.

Use HSM to store packager credentials

Primetype Offline Packager supports access to PackagerCredential certificates (required for AAXS-based content protection) and common keys stored in a hardware security module (HSM) for enhanced security.

To enable access to either packager credentials or common keys or both in the HSM, you require a SunPKCS11 configuration file and an HSM password.

The SunPKCS11 configuration file contains the following tags:

| Tag | Description |
|---|--|
| <code><hsm_sunpkcs11conf_file>hsm_conf/pkcs11.cfg</hsm_sunpkcs11conf_file></code> | The 'pkcs11.cfg' config file used with the SunPKCS11 provider to access HSM Module |
| <code><hsm_pwd></hsm_pwd></code> | Password in Base64 format to access your partition under HSM |
| <code><use_hsm_pkgr_pfx/></code> | Specifies that the Offline Packager should access packager credential certificate from HSM |
| <code><pkgr_pfx>drmPackagerCredentialAlias</pkgr_pfx></code> | Specifies the alias for packager credential certificate |
| <code><use_hsm_common_key/></code> | Specifies that the Offline Packager should access common key from HSM Module |
| <code><common_key_file>commonKeyAlias</common_key_file></code> | Specifies the alias for common key |

The following is a sample configuration for the SunPKCS11 configuration file:

```
<use_hsm_pkgr_pfx/>
<pkgr_pfx>drmPackagerCredentialAlias</pkgr_pfx>
<use_hsm_common_key/>
<common_key_file>commonKeyAlias</common_key_file>
<hsm_sunpkcs11conf_file>hsm_conf/pkcs11.cfg</hsm_sunpkcs11conf_file>
<hsm_pwd>VxfVTnUVjMn=</hsm_pwd>
```

The following is the complete configuration for the SunPKCS11 configuration file:

```
<config>
  <in_path>/home/Testdata/myFile.ts</in_path>
  <out_type>hls</out_type>
  <out_path>output/HLS_AAXS_HSM</out_path>
  <name>hls_sample_hsm_protected</name>
  <drm/>
  <drm_sys>AAXS</drm_sys>
  <frag_dur>4</frag_dur>
  <target_dur>6</target_dur>
  <lic_svr_url>http://server_ip:port</lic_svr_url>
  <lic_svr_cer>creds/static/license_certificate</lic_svr_cer>
  <lic_svr_pfx>creds/static/license_certificate_credential</lic_svr_pfx>
  <lic_svr_pfx_pwd>password</lic_svr_pfx_pwd>
  <use_hsm_pkgr_pfx/>
  <pkgr_pfx>drmPackagerCredentialAlias</pkgr_pfx>
  <use_hsm_common_key/>
  <common_key_file>commonKeyAlias</common_key_file>
  <hsm_sunpkcs11conf_file>hsm_conf/pkcs11.cfg</hsm_sunpkcs11conf_file>
  <hsm_pwd>password</hsm_pwd>
  <transport_cer>creds/static/transport_certificate</transport_cer>
  <policy_file>creds/static/policy.pol</policy_file>
  <content_id>HLS_AAXS_HSM</content_id>
</config>
```

To enable HSM access from Live Packager, configure the `<HSMModule>` element in the `server.xml` file within the `<ContentProtection>/<FAXS4>` element.

The following is a sample configuration for the `<ContentProtection>` element with HSM access enabled:

```
<ContentProtection>
  <FAXS4>
    ...
    <HSMModule>
      <!-- Provide Config information for the HSM Module access -->
      <SunPKCS11ConfigFile></SunPKCS11ConfigFile>
      <!-- Config file used with the SunPKCS11 Provider to access HSM -->
      <Password></Password>
      <!-- base64 encoded format -->
      </HSMModule>
    ...
  </FAXS4>
</ContentProtection>
```

The `useHSM` attribute for the `PackagerCredential` element indicates if the credential should be accessed using HSM or the file system. The credential is an alias if it is accessed using HSM.

```
<PackagerCredential useHSM="true">...</PackagerCredential>
```

The `useHSM` attribute for the `CommonKey` element indicates whether the key should be accessed using HSM or the file system.

```
<CommonKey useHSM="true">...</CommonKey>
```

The following is the complete configuration for the Content Protection element:

```
<Config>
  <ContentProtection>
    <FAXS4>
      <HSMModule>
        <SunPKCS11ConfigFile>hsm_conf/pkcs11.cfg</SunPKCS11ConfigFile>
        <Password>VxfVTnUVjMn=</Password>
      </HSMModule>
      <LicenseServerURL>http://primetype.aaxs.adobe.com</LicenseServerURL>
    </FAXS4>
  </ContentProtection>
  <LicenseServerCertificate>creds/static/phds_license_server.der</LicenseServerCertificate>
  <LicenseServerCredential>creds/static/phds_license_server.pfx</LicenseServerCredential>
  <LicenseServerCredentialPassword>v/lB4EoNQqQ=</LicenseServerCredentialPassword>
  <PackagerCredential useHSM="true">packager-credential-alias</PackagerCredential>
  <TransportCertificate>creds/static/phds_production_transport.der</TransportCertificate>
  <CommonKey useHSM="true">commonkey-alias</CommonKey>
  <PolicyFile>creds/static/phds_policy.pol</PolicyFile>
  <RecipientCertificates>creds/sd</RecipientCertificates>
</Config>
```

Generate a trick play stream

The packager supports the trick play functionality in the player, by generating a key-frame only stream.

To generate such a stream, add the `kfonly` tag in the configuration file or command line. The stream contains key-frames only. Each trickplay segment contains only the first keyframe and all other frames are dropped. Only video `AdaptationSet` is generated for `kfonly` packaging.

To include I-frame only streams in HLS set level playlist, do the following steps:

1. Use Offline Packager with the `kfonly` parameter, to generate a key frame only stream for trick play. The properties file created for I-frame only stream has `keyFrameOnly` parameter set to `True`.
2. Using the `in_path` parameter, provide the m3u8 file with the I-frame only stream, along with the other m3u8 files, while executing `create_playlist` command. This includes `EXT-X-I-FRAME-STREAM-INF` in the output m3u8 file.

For more information on the commands, see [Configuration parameters](#).

An example of the configuration file

```
<config>
<in_path>input.mp4</in_path>
<out_type>hds</out_type>
<out_path>kfonly_hds</out_path>
<kfonly/>
</config>
```

Trick play is supported for DASH. The following is a sample MPD file generated by Offline Packager using `kfonly` flag, `frag_dur=4`, and `out_type=dash` parameters.

```
<?xml version="1.0" encoding="UTF-8"?><MPD xmlns="urn:mpeg:DASH:schema:MPD:2011"
xmlns:scte35="urn:scte:scte35:2013" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
mediaPresentationDuration="PT298S" minBufferTime="PT4S"
profiles="urn:mpeg:dash:profile:isoff-live:2011" type="static"
xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 DASH-MPD.xsd">
<Period id="1" start="PT0S">
<AdaptationSet bitstreamSwitching="true" contentType="video" id="3" mimeType="video/mp4"
segmentAlignment="true" startWithSAP="1">
<Representation bandwidth="25000" codecs="avc1.42c01e" codingDependency="false" frameRate="1/4"
height="360" id="1" maxPlayoutRate="100" width="480">
<BaseURL>trickmode</BaseURL>
<SegmentTemplate initialization="sInit.mp4" media="s$Number$.m4s" startNumber="0"
timescale="1000">
<SegmentTimeline>
<S d="4000" r="73" t="0"/>
<S d="2075" t="296000"/>
</SegmentTimeline>
</SegmentTemplate>
</Representation>
<EssentialProperty schemeIdUri="http://dashif.org/guidelines/trickmode" value="1"/>
</AdaptationSet>
</Period>
</MPD>
```

Closed Captioning (CC) support

Closed captioning lets the content provider display text overlay on video content to help hearing impaired users understand the content. Closed captioning is primarily used as an accessibility feature to represent the audio portion of the video content (and any additional information necessary to understand the content such as speaker identification) as it occurs. The term "closed" in closed captioning indicates that the captions can be turned on or off by the viewer. This feature requires that the video player contain the capability to decode and render the captioning data.

The captioning data embedded in the stream can be playable by clients supporting the Primetime Player SDK. Packager extracts the 608/708 captioning data from the MPEG2TS source stream, wraps the captioning data in a special AMF0 encoded data message called "onCaptionInfo" message, and embeds the message into the stream.

The input format for closed captioning is H264 as SEI NALUs for both mp4 and TS. For HDS output, corresponding onCaptionInfo AMF message will be inserted in HDS output. For HLS output, caption data present in H264 bytes will be simply passed through.

Packaging WebVTT files for HLS delivery

Primetype OfflinePackager supports repackaging of WebVTT and SRT files for HLS delivery. WebVTT files are used to mark up external text track resources for your video files. You can create separate WebVTT streams (m3u8/VTT fragments) from an input WebVTT or SRT file by specifying it in the `in_path` parameter.

The following is an example of how an WebVTT stream can be created:

```
java -jar OfflinePackager.jar -in_path ~/vtt/orig/english_en_full.vtt -out_path
~/webroot/vtt/orig -out_type hls
```

The output of the Offline Packager, for the source WebVTT or SRT file, is available at

`/webroot/vtt/orig/english_en_full.vtt`.

The m3u8 for the VTT stream is shown below:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:4
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-MEDIA-TIME:12.0
#EXTINF:4,
english_en.12000.vtt
#EXTINF:4,
english_en.16000.vtt
#EXTINF:4,
english_en.20000.vtt
#EXTINF:4,
english_en.24000.vtt
#EXTINF:4,
english_en.28000.vtt
#EXTINF:4,
..
#EXTINF:4,
english_en.140000.vtt
#EXT-X-ENDLIST
```

The following is an example of how a VTT stream can be included in the HLS master playlist:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="English",DEFAULT=YES,URI="/vtt/orig/english_en_full.m3u8",LANGUAGE="en"

#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=150000,SUBTITLES="subs"
/vtt/video/movie.m3u8
```

You can include multiple sources of text tracks say, to provide support for subtitles in multiple languages. You cannot give multiple files as input, so individually package each VTT or SRT file.

1. Package the main media.
2. Package the individual VTT or SRT files.
3. Run the PlaylistCreator tool to create a variant playlist for the media files.
4. To use the PlaylistCreator tool to associate VTT files, use the tool with the type identifier as 's' in the association parameter. For more details, see [PlaylistCreator tool](#).

An example with two VTT files in the variant playlist is:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="English",
```

```

DEFAULT=YES, FORCED=NO, URI="vtt/english.m3u8", LANGUAGE="en"
#EXT-X-MEDIA:TYPE=SUBTITLES, GROUP-ID="subs", NAME=" ",
DEFAULT=NO, FORCED=NO, URI="vtt/chinese.m3u8", LANGUAGE="zh-Hans"

#EXT-X-STREAM-INF: BANDWIDTH=2670000, CODECS="mp4a.40.2, avc1.4d401f",
RESOLUTION=640x360, SUBTITLES="subs" 2.5Mbps/prog_index.m3u8

```



Note: In `EXT-X-STREAM-INF` tag, the value of subtitles must match the value of the `GROUP-ID` attribute of an `EXT-X-MEDIA` tag in the variant playlist whose `TYPE` attribute is `SUBTITLES`.

Providing Closed Captions for MPEG-DASH delivery

Closed captions, for MPEG DASH packaging, are provided primarily in WebVTT format. SEI based CEA-608/708 closed captions, embedded in the source video stream, are added to the video segments but not signalled at the MPD level.

To package the closed captions in source WebVTT or SRT file, provided in `in_path`, it must have the extension `.vtt` or `.srt`. The language of the closed captions can be specified optionally using `lang` parameter. There is a single file containing all closed captions in the Representation and multiple segments are not generated as illustrated in the following sample:

```
java -jar OfflinePackager.jar -in_path cc.vtt -out_type dash -out_path dash_output -frag_dur 4 -lang es
```

Sample of an `AdaptationSet` containing WebVTT closed captions generated by the above is:

```

<AdaptationSet id="4" lang="es" mimeType="text/vtt">
<Representation id="1">
<BaseURL>cc.vtt</BaseURL>
</Representation>
</AdaptationSet>

```

Still Image support for audio-only stream

Adobe Primerime Offline Packager supports embedding a still image with an audio-only stream.

An image can be added to an audio-only HLS segment. If a HLS player switches to an audio-only stream in case of a low bandwidth network condition, it displays the embedded still-image. Using a still image in the audio stream, a content publisher can convey a message, serve album art or poster, or inform about low bandwidth condition. A still image can be embedded in an audio-video stream but is ignored during playback.

To embed an image, specify the path of the image file using the Offline Packager's `-image_file` option. For an understanding of the options, see [Configuration parameters](#).

```
java -jar OfflinePackager.jar -out_path "C:\hls_audio_only" -out_type hls -track_ids 2 -in_path "C:\Data\sample.mp4" -image_file "C:\stillimage.png"
```

Apple Safari on iOS7 supports playing still images with an audio-only stream. Some image specifications that are recommended are:

- JPG image; 30kb or lesser in size and 290 x 260 pixels or lesser in dimensions.
- PNG image; 30kb or lesser in size and 400 x 200 pixels or lesser in dimensions.

Inserting advertisement cues

The Offline Packager can prepare the stream for Ad insertion by providing the position and optionally the duration of the Ad break. The generated HLS files are aligned based on the splicing boundaries, so that the advertisements can be inserted without the re-segmentation of the stream.

1. Create an external file that contains the slicing details.
2. Run Offline Packager using the additional parameter, `cue_info`, with the external file as the value.
Specify the following parameters in an external file (cue information file) as comma-separated values:
 - Id: The ID of the advertisement cue. It can be a string or a number.
 - Splice time: The splice time in seconds on which the splicing boundary needs to be inserted.
 - Splice duration (optional): If this value is specified, the content of the stream is replaced by an advertisement of the same duration.

Default value of this parameter is 0 (if it is not specified). If splice duration is not specified, the advertisement inserted doesn't replace the main content in the stream.

A sample cue information file:

```
a1, 10.5  
a2, 33.3, 10  
a3, 77.8, 0
```

In the sample file above, the second splice point, `a2`, replaces the content in the stream for 10 seconds.

Alternatively, you can specify external Ad cues through command line using the `-cues` option. The following is an example where Ad cues are provided using the command line:

```
java -jar OfflinePackager.jar -in_path ~/Test/Files/video_360p.mp4 -out_path ~/Test/Output  
-out_type hls -cues 1,0;2,10,10;3,30,15 -add_cue_info
```



Note:

The Unix/Linux Shell uses ';' as a delimiter. If you specify ad cues using the command line, enclose the semicolons within quotes to prevent them from being interpreted as delimiters by the Shell.

Advertisement cues for HLS

If the `add_cue_info` parameter is present, the generated M3U8 file contains the `#EXT-X-CUE` custom tag associated with URI of the segment which starts at this cue-time or the splice point. If a splice duration is specified (content needs to be replaced), the main content segment URIs, that lies in the advertisement break, are associated with the `#EXT-X-CUE-CONT` tag.

A sample M3U8 file generated for the cue information file provided above is given below:

```
#EXTM3U  
#EXT-X-VERSION:3  
#EXT-X-TARGETDURATION:7  
#EXT-X-PLAYLIST-TYPE:VOD  
#EXTINF:4.004, sample0.ts  
#EXTINF:4.004, sample4004.ts  
#EXTINF:2.502, sample8008.ts  
#EXT-X-CUE:TYPE=SpliceOut,ID=a1,TIME=10.5  
#EXTINF:1.502, sample10510.ts
```

```

...
#EXT-X-CUE:TYPE=SpliceOut,ID=a2,TIME=33.3,DURATION=10.0
#EXTINF:2.735, sample33301.ts
#EXT-X-CUE-CONT:ID=a2
#EXTINF:4.004, sample36036.ts
#EXT-X-CUE-CONT:ID=a2
#EXTINF:3.266, sample40040.ts
#EXTINF:0.738, sample43306.ts
...
#EXT-X-CUE:TYPE=SpliceOut,ID=a3,TIME=77.8
....

```

| Attribute | Type | Required | Description | Source field of SCTE35 message |
|-----------------|---------|----------|--|--------------------------------|
| TYPE | String | Required | The event type which must be "SpliceOut" or "SpliceIn" | out_of_network_indicator |
| ID | String | Required | A unique identifier for this event within the context of the program stream. | splice_event_id |
| TIME | Number | Required | The stream's presentation time in fractional seconds at which point the splice out should occur. | pts_time + pts_adjustment |
| DURATION | Number | Required | The splice duration in fractional seconds. A value of "0" means the duration is not known. | duration (in break_duration()) |
| PROGRAM-ID | Integer | Optional | Identifier for the program content this splice applies to. | unique_program_id |
| AVAIL-NUM | Integer | Optional | Index of this specific avail within the total set of avails within the viewing event (identified by PROGRAM-ID). | avail_num |
| AVAILS-EXPECTED | Integer | Optional | The expected total number of avails within the viewing event (identified by PROGRAM-ID). | avails_expected |

Advertisement cues for HDS

For HDS output, the HDS fragments are not split at the splice points. The manifest file (F4M) has the <cue> tags inside <cueInfo> for each cue.

Sample manifest file:

```

<manifest xmlns="http://ns.adobe.com/f4m/1.0">
...
<cueInfo id="c1234">
<cue id="a1" time="10.5" type="SpliceOut"/>

```

```
<cue duration="10.0" id="a2" time="33.3" type="SpliceOut"/>
<cue id="a3" time="77.8" type="SpliceOut"/>
</cueInfo>
...
</manifest>
```

The following cue points are dropped:

- A cue that overlaps the preceding cue.
- A cue with a negative duration.

A sample configuration file for HDS:

```
<config>
<in_path>turner400_2min.ts</in_path>
<out_type>hds</out_type>
<out_path>cc_drmRefresh_aes128_Cues_output\embedded_cue_ts_hds</out_path>
<frag_dur>4</frag_dur>
<target_dur>6</target_dur>
<cue_info>cue_info_file_embedded.csv</cue_info>
<add_cue_info/>
</config>
```

Advertisement cues for MPEG-DASH

DASH segment boundaries are aligned with the Ad cues provided either in-stream or externally in a file or via command line or via configuration. Segment boundary alignment at cue-points facilitates client-side and server-side Ad insertion later via insertion of Period(s).

The cues are added in MPD only when `add_cue_info` is set. Using this option is not recommended, because the cues, in the MPD, are not processed by the player.

Insert the embedded cue information

The advertisement cue present in input stream can be used. If cue information is provided in the external file and internally, the externally provided cues will be processed with higher priority, if they overlap with any in-stream cues.

Use the `in_stream_cue_info` parameter to process the embedded cue information.

Sample configuration file:

```
<config>
<in_path>turner400_2min.ts</in_path>
<out_type>hds</out_type>
<out_path>cc_drmRefresh_aes128_Cues_output\embedded_cue_ts_hds</out_path>
<frag_dur>4</frag_dur>
<target_dur>6</target_dur>
<cue_info>cue_info_file_embedded.csv</cue_info>
<add_cue_info/>
<in_stream_cue_info>
</config>
```

Ad cues in command line

You can specify external Ad cues through the command line or in the configuration-file using the `-cues` option in Primetime Offline Packager. The following is an example where cues are provided in the command line:

```
java -jar OfflinePackager.jar -in_path ~/Test/Files/video_360p.mp4 -out_path ~/Test/Output
-out_type hls -cues 1,0;2,10,10;3,30,15 -add_cue_info
```



Note:

Shell uses ';' to separate commands, so when the list of ad cues is provided on the command line it must be quoted to avoid semicolon interpretation by shell.

Align Ad Cues with keyframes

For HLS, cues should align with Keyframes. In scenarios where exact keyframe time is not known, Offline packaging can be done with an option to splice at the next available keyframe after the cue point. The optional parameter `splice_at_next_kf` has been added to facilitate this. This is an optional parameter but should be given if the specified cue points are not exactly at keyframes. Setting it to true will jump to next available keyframe after the specified cue point. The default value is false.

Use of `splice_at_next_kf` is not recommended for DASH output where audio and video are not multiplexed and segmented independently, because audio segments get aligned with the provided cues and not adjusted according to the keyframe-occurrences.

SCTE cues

The Primetime Offline Packager can parse SCTE 35 cues from input ts file and add them as cue information to f4m for HDS output and m3u8 for HLS output.

OfflinePackager has can parse in-stream SCTE 35 cues and add them as cue information in the generated manifest. The Primetime OfflinePackager captures SCTE 35 cues from input ts/mp4 file and passes them along so that they can be used for advertising decisioning/insertion/ blackouts by downstream processors. These cues are added in f4m for HDS output and m3u8 for HLS output.

SCTE 35 cue parsing supports a wider part of SCTE 35 specifications. The Offline Packager generates cues in formats described in the Digital Program Insertion (DPI) specifications.

Cue Signal Format for manifest files

Primetime Offline Package lets you add cue information in m3u8 and f4m manifest files in the format specified by the [Primetime Digital Program Insertion Signaling specification](#).

You can determine the format of cue information to be added in the f4m or m3u8 using the command line option `cue_mode`. The following are the valid values for this command line option:

| | |
|--------|--|
| PT_1_0 | To generate cue information in PT 1.0 format as described in Primetime HLS Ad cue format specifications for m3u8 and Adobe Media Manifest (F4M) Specification (Version 3.0) for f4m. |
|--------|--|

| | |
|------------|--|
| DPI Simple | To generate cue information in DPI Simple mode as described in the Primetime Digital Program Insertion Signaling specification . |
| DPI SCTE35 | To generate cue information in DPI SCTE mode as described in the Primetime Digital Program Insertion Signaling specification . |

The DPISCTE35 mode is used if no cue mode is specified in the command line option.

Supported SCTE-35 splice_insert() messages

Restrictions for the set of supported SCTE-35 splice_insert() messages.

- The SCTE35 message must be splice_insert()

The value of splice_command_type in splice_info_section() message will be 0x5.

- out_of_network_indicator

At present, the Primetime Packager will only process messages where this flag is 1 which indicates a Splice Out point. Any SCTE-35 splice-in messages (out-of-network = 0) are ignored.

- duration_flag must be set.

If a break_duration() is not defined, the message will be ignored.

- The value of auto_return flag in a break_duration() is ignored

It is assumed that an auto- return from insertion stream content to the network feed should happen at the end of the break duration.

- Messages having the splice_immediate_flag = 1 and no defined splice_time() are ignored

An explicit splice_time() needs to be defined. The value of program_splice_flag must be 1.

- splice_event_cancel_indicator

splice_insert() messages with splice_event_cancel_indicator set to true are currently ignored.



Configuration parameters

| Parameter Name | Description | Data Type | Type |
|-------------------|--|-----------|----------|
| aaxs_external_cek | Allows packaging content with AAXS DRM and create metadata that does not contain the CEK. | Boolean | Optional |
| add_cue_info | Specifies to add the advertisement cues to the manifest files of HDS and HLS output. This needs to be specified with cue_info or in_stream_cue_info options. | NA | Optional |

| Parameter Name | Description | Data Type | Type |
|---------------------------|--|--------------|--|
| bit_rates | Comma-separated list of content bitrate. It is specified in kilobits per second. This needs to be specified for each file mentioned in the in_path parameter. This option is needed only for multi-bit-rate packaging. | Integer list | Optional |
| buydrm_contentid | BuyDRM content identifier in GUID format. If it is not specified, then a new ContentID is randomly generated. | String | Optional |
| buydrm_keyid (deprecated) | BuyDRM Key identifier in GUID. This configuration is deprecated; use key_id instead. | String | Mandatory when drm_sys is set to playready_buydrm. |
| buydrm_mediaid | BuyDRM media identifier that uniquely identifies media file. If it is not provided, then it defaults to the source file specified in in_path. | String | Optional |
| buydrm_proxy_url | The URL of BuyDRM proxy server. If it is provided, then it is set in the Playready Header Object instead of the license server URL. | String | Optional |
| buydrm_userkey | BuyDRM User Key in GUID format | String | Mandatory when drm_sys is set to playready_buydrm |
| common_key_file | The common key file path. | String | Required for AAXS DRM. |
| content_id | Content ID that needs to be used for the key generation for content protection. | String | Required for AAXS DRM. |
| cue_info | Specifies the file that contains the advertisement cues. | String | Optional |
| cue_mode | Cue information signaling mode to be used for manifest. Allowed values are PT_1_0/DPISimple/DPIStc35 | String | Optional. If not specified, default value is DPIStc35. |
| cues | Semicolon separated list of ad cues | String | Optional |
| drm | Enables the content protection. By default, content protection is disabled. | NA | Optional |
| drm_refresh | Regenerates the DRM-metadata without repackaging the contents. It is a flag type parameter. | NA | |

| Parameter Name | Description | Data Type | Type |
|------------------------|---|-----------|--|
| drm_sys | Specifies the DRM system that needs to be used for content protection. It can be AAXS, playready_buydrm, FairPlay, Widevine, or none. Specify as none, if you want simple protection without using DRM for HLS output. The default value of DRM system is AAXS. For multi-DRM case, <code>drm_sys</code> is a list of comma separated applicable values. For instance, <code>fairplay,aaxs</code> for HLS, or <code>playready,widevine,aaxs</code> for DASH. | String | Optional |
| encrypt_audio | Specifies whether to encrypt the audio in the file. Default value is <code>true</code> . It's applicable only if the output type is HDS. | String | Optional. Applicable only for HDS output type. Ignored for other output types. |
| encrypt_data | Specifies whether to encrypt the data in the file. Default value is <code>true</code> . It's applicable only if the output type is HDS. | String | Optional. Applicable only for HDS output type. Ignored for other output types. |
| encrypt_video | Specifies whether to encrypt the video in the file. Default value is <code>true</code> . It's applicable only if the output type is HDS. | String | Optional. Applicable only for HDS output type. Ignored for other output types. |
| ext_bs | This flag enables generation of an external bootstrap for the HDS outputs. It is a deprecated option. | String | Optional |
| frag_dur | Fragment duration in seconds. For empty and invalid input, four seconds is the default value. | Float | Optional |
| help | Provides the usage and version information. | NA | Optional |
| hls_enc_type | The acceptable value is <code>sample</code> , which indicates Sample-AES encryption. If this parameter is not specified then encryption is of type AES-128. | String | Optional. Mandatory for Sample-AES encryption. |
| hsm_pwd | Scrambled Password that lets you access HSM Module. To scramble the password, Base64 encode it. | String | Partition password for the HSM Module |
| hsm_sunpkcs11conf_file | Path of the configuration file used with the SunPKCS11 Provider to access HSM. | String | Specify this file path if <code>use_hsm_pkggr_pfx</code> or |

| Parameter Name | Description | Data Type | Type |
|--------------------|---|-----------|--|
| | | | use_hsm_common_key flag is added. |
| image_file | Path of a still image that is embedded in a HLS audio-only stream. | String | Optional. Required only for embedding still image in HLS audio-only steam. |
| in_path | Path of the MP4 or TS input files. Provide a comma separated list of input files for multi-bitrate packaging. For DRM refresh, specify the <code>drm_refresh</code> tag. The <code>in_path</code> argument is relative to the command line location, so use an absolute path if necessary. | String | Optional |
| in_stream_cue_info | Specifies to add advertisement cues embedded in the input file provided. | NA | Optional |
| info | Displays information of the source content, such as the stream tracks contained. It is a flag type parameter. | NA | Optional |
| inline_drm | inline_drm is applicable only for AAXS and its default value is false. This parameter places DRM-metadata inline in the manifest. | NA | Optional |
| iv_out_path | Path of the file to output the auto-generated IV. The output file contains the hex encoded packager generated IV. If this parameter is not specified and <code>log_iv</code> flag is enabled, then the IV is written to the content output directory. | String | Optional |
| iv_file_name | Name of the file containing the packager-generated IV. It's an optional parameter. It should only be provided when the packager generated IV is used. If this parameter is not provided when the packager-generated IV is used, then the IV file is named <code>IV.txt</code> . If a custom IV is provided, this option is redundant and hence ignored. | String | Optional |
| iv_file_path | Path of the file containing the hex encoded, user-generated IV for AES-128 encryption of HLS output. It is applicable only for HLS output. It's an optional parameter. If this is not provided, a random IV is generated by the Offline Packager. | String | Optional |
| key_base_url | Packager-generated key to be used for vanilla HLS encryption. | String | Optional |

| Parameter Name | Description | Data Type | Type |
|----------------|---|-----------|---|
| key_file_name | Packager-generated key to be used for vanilla HLS encryption. | String | Optional |
| key_file_path | This file contains the path of the user-generated Base 64 encoded AES-128 and Sample-AES keys. If it is not specified, the Offline Packager generates the key. If key file path is specified, then the Offline Packager uses the specified key instead of the one derived from contentID and common-key. | String | Optional |
| key_id | A 32 character Hex string used in DASH content protection. | String | Mandatory for Playready, Widevine, AAXS DRM with DASH, and AAXS DRM for HLS with external CEK support. Use this parameter instead of <code>widevine_key_id</code> and <code>playready_keyid</code> that are deprecated. |
| key_out_path | The directory where packager-generated HLS key file should be copied. If not specified, key file generated is copied to the <code>out_path</code> specified. When packaging multi-bit rate encrypted content, if <code>key_out_path</code> is configured, the keys are written inside individual bitrate folders at <code>key_out_path</code> path. | String | Optional |
| key_rot | Enables key rotation.  Note: Key rotation is only applicable to AAXS in HLS. | NA | Optional |
| key_rot_dur | Specifies the key rotation interval in seconds. The default interval is 900 seconds.  Note: Key rotation is only applicable to AAXS in HLS. | Integer | Optional |
| key_url | The key server URL is used for HLS output with AAXS, vanilla AES-128 encryption, or Sample AES. For FairPlay, <code>key_url</code> is copied in the URI parameter of the EXT-X-KEY tag of the m3u8 file. For Fairplay, <code>key_url</code> is a mandatory parameter. | String | Mandatory for FairPlay, else it is optional. |

| Parameter Name | Description | Data Type | Type |
|------------------------|---|-----------|--|
| kfonly | Generates a key frame-only stream for trick play. | NA | Optional |
| lic_svr_cer | The license server certificate path. | String | Required for AAXS DRM. |
| lic_svr_pfx | The license server credentials path. This optional parameter is required only if type of license in policy is Enhanced License Chaining or PHDS/PHLS. | String | Required for AAXS DRM. |
| lic_svr_pfx_pwd | The license server credentials password. This optional parameter is required only if type of license in policy is Enhanced License Chaining or PHDS/PHLS. | String | Required for AAXS DRM. |
| lic_svr_url | The license server URL. | String | Required for AAXS DRM when policy requires license server. |
| log_iv | When this flag is added, the Offline Packager writes the packager-generated IV to a file. This flag is not applicable for a custom IV provided by the user. For packager-generated IV, IV is not written to external file if this flag is not provided. | Flag | |
| ms_unencrypted | The duration in milliseconds at the beginning of the content that remains unencrypted. Use this option to improve stream start time, because it allows the client to acquire the license asynchronously as playback begins. Default value is 0. | Integer | Optional |
| out_path | The output directory path. | String | Mandatory |
| on_demand_dash_profile | Use this parameter to package DASH content conforming to the on-demand profile. | Boolean | Available only if <code>out_type</code> is DASH. |
| out_type | The output format types. It can be HDS, HLS, or DASH. | String | Mandatory |
| pkgr_pfx | The packager credentials path. | String | Required for AAXS DRM. |
| pkgr_pfx_pwd | The packager credentials password. | String | Required for AAXS DRM. |
| policy_file | Policy file path. | String | Required for AAXS DRM. |
| rec_cer | Recipient certificates directory path. By default, the recipient certificates are packaged with Offline Packager in the | String | By default recipient certificates are packaged |

| Parameter Name | Description | Data Type | Type |
|--------------------|---|--------------|--|
| | install_dir/cred/sd folder. If you don't specify any recipient certification path, certificates are obtained from the default path. | | with the Offline Packager in cred/sd folder. If a user doesn't specify any recipient certs path specifically, certificates are picked from default path. |
| splice_at_next_kf | This is an optional parameter but should be given if the specified cue points are not exactly at keyframes. Setting it to true will jump to next available keyframe after the specified cue point. | Flag | Optional. Default value is false. |
| target_dur | The target duration for HLS output. If it is not specified, the target duration is set as the longest fragment's duration. If the longest fragment duration is higher than the target duration specified, the longest fragment duration is used as the target duration. | Integer | Optional |
| top_level_prefix | Filename prefix for the manifest file (f4m) or variant playlist (m3u8.) This can be used for multi-bitrate packaging. Default is set.f4m for HDS output and variant.m3u8 for HLS output. | String | Optional |
| track_ids | A comma-separated list of format-specific track identifiers. This is used for packaging specific tracks. | Integer list | Optional |
| transport_cer | The transport certificate path. | String | Required for AAXS DRM. |
| use_hsm_common_key | Add this flag to indicate that the key should be accessed using the HSM Module. If this flag is enabled then the common key param is actually an alias. | Flag | Optional |
| use_hsm_pkgr_pfx | Indicates that the packager credential should be accessed using the HSM Module. If this flag is enabled, the packager credential is an alias. | Flag | Optional |
| validate | Validates the configuration file provided. If specified, only validation is performed. Packaging is omitted. | NA | Optional, could be specified together with cue_info or in_stream_cue_info |
| verbosity | Sets the verbosity level. Verbosity level can be 1 to 3. Three is the most verbose and the default value. | Integer | Optional |

| Parameter Name | Description | Data Type | Type |
|--|--|-----------|--|
| <code>widevine_content_id</code> | A content identifier as specified by the content provider. The ID must be Hex encoded. | String | Mandatory along with <code>widevine_provider</code> if <code>widevine_header</code> is not provided. |
| <code>widevine_header</code> | Widevine Base 64 encoded header for use in PSSH box. | String | Mandatory if <code>widevine_content_id</code> and <code>widevine_provider</code> are not specified. |
| <code>widevine_key_id</code> (deprecated) | Widevine Hex encoded encryption key identifier. This configuration is deprecated; use <code>key_id</code> instead. | String | Mandatory |
| <code>widevine_provider</code> | The name of the content provider. | String | Mandatory along with <code>widevine_content_id</code> if <code>widevine_header</code> is not provided. |
| <code>whitelist</code> | Specifies the folder that contains the app whitelist and/or SWF verification information. It can be specified only if the policy for content protection is PHDS or PHLS. | String | Optional |