HLS

HTTP Live Streaming

While Bento4 is an MP4 file format library, and HLS uses the MPEG2 TS (Transport Stream) format, Bento4 includes a set of tools and functions that allow the conversion from MP4 to MPEG2 TS, as well as tools to create the .m3u8 playlists for HLS (HTTP Live Streaming).

Specifications

The specification for HTTP Live Streaming (HLS) can be found in an IETF Draft: HTTP Live Streaming – draft-pantos-http-live-streaming

Quick introduction to HLS

An HLS presentation consists of media segments that contain audio, video, subtitles, or a multiplex of both audio and video. Media segment URLs are referenced from media playlists (a text-based format, usually stored in files with the `.m3u8` file extension).
When an HLS presentation has multiple bitrate variants for the same content (to allow adaptive streaming), there is a media playlist for each bitrate variant, and a *master playlist* that points to the individual media playlists (instead of referencing media segments).
Preparing content for HLS streaming invovles creating the `MPEG2-TS` media files and the playlists.
Media segments can be stored either as discrete files (one file per segment), or together in a single file (in which case the client will access the segment using HTTP Range requests). Optionally, the media files may be encrypted, according to one of several supported encryption modes (`AES-128` and `SAMPLE-AES`).

Tools

`mp4hls`

This is a high-level tool, which uses `mp42hls` as a helper tool. It creates a multi-bitrate HLS master playlist from one or more MP4 files, including support for encryption, I-frame-only playlists, and subtitles. This tool uses the 'mp42hls' low level tool internally, so all the options supported by that low level tool are also available. This can be used as a replacement for Apple's `variantplaylistcreator` tool.
The *Using `mp4hls`* section below is a high level guide to this tool. For a mode detailed list of all the command line options, visit the `mp4hls` tool documenation page.

`mp42hls`

This is a low-level tool that converts an MP4 file to a single-bitrate HLS (HTTP Live Streaming) presentation, including the generation of the segments and .m3u8 playlist as well as optional `AES-128` and `SAMPLE-AES` (for Fairplay) encryption. This can be used as a replacement for Apple's `mediafilesegmenter` tool.
Visit the `mp42hls` tool documenation page for details.

Using `mp4hls`

In its simplest usage, `mp4hls` takes one or more MP4 files as input and produces a complete HLS presentation in a directory named `output`. That includes the media segments as well as the media playlist(s) and master playlist.
By default, `mp4hls` tries to be conservative and not assume that the HLS client implements all the newer optional functionality of the more recent versions of the specification.
The default HLS version used is 3. As a result, the default mode is to use a separate file for each media segment. This can lead to a presentation with a lot of small files. To create instead an HLS presentation with a single file per bitrate (in which case the HLS client will use HTTP Range requests to access the segments), use the `--output-single-file` command line option.

Examples
HLS presentation with a single multiplexed audio/video file.

```
mp4hls video.mp4
```

This will create a directory named `output` with all the media segment files, media playlists and master playlist (named `master.m3u8` by default)

HLS presentation with multiple bitrates, using a single media file per bitrate

```
mp4hls --output-single-file video_00500.mp4  video_00800.mp4 video_01100.mp4
```

I-Frame-only playlists

`mp42hls` supports the generation of I-Frame-only media and playlists. Since this feature requires an HLS version of 4 or greater, you must use the command line option `--hls-version=4` (or higher).

Encryption and DRM

`mp4hls` can encrypt the media segments and include encryption info in the media playlists.
There are two types of encryption that are supported:

- segment-based `AES-128` encryption. In that mode, each media segment is encrypted using `AES-128` in CBC mode. This works both with discrete media segment files as well as with single-file (`--output-single-file` option). This is the default encryption mode.
- sample-based `SAMPLE-AES` encryption. In that mode, instead of encrypting entire media segments in bulk, audio and video samples are encrypted at a lower level (audio frames for audio and video NAL units for video). To select this encryption mode, use the `--encryption-mode=SAMPLE-AES` command line option.

Regardless of the encryption mode, the encryption key may be delivered to the client in one of multiple ways: Simple HTTP Key Delivery, FairPlay Streaming, or a custom (other DRM) method.

Simple HTTP Key Delivery

With Simple HTTP Key Delivery, media playlists contain the URL to the key. The key is simply a 16 byte binary payload.
To use this mode of key delivery, you specify the encryption key using the `--encryption-key` command line option.
If you want to use a key URL different from the default (`key.bin`), you can specify that URL with the `--encryption-key-uri` command line option.
You can also use the `--output-encryption-key` option to tell the tool to save the key in a file named `key.bin`, which will then be visible to the HTTP client through the default encryption key URL.

Example
Encrypted content, with the key stored in the file `key.bin` and accessible through an HTTP URL

```
mp4hls --encryption-key baab6d0dd153762d945d5a060abb5fcd --output-encryption-key video.mp4
```

FairPlay Streaming

With FairPlay streaming, the FairPlay DRM system is used to deliver both a 16-byte key and a 16-byte IV. FairPlay only works in combination with `SAMPLE-AES` encryption.
In order to use FairPlay streaming key delivery, you must:

- use the command line option `--encryption-key=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY` to specify the encryption key (Xs, in hex) and IV (Ys, in hex)
- use the command line option `--encryption-mode=SAMPLE-AES` to select the `SAMPLE-AES` encryption mode
- use the command line option `--encryption-key-format=com.apple.streamingkeydelivery` to specify that FairPlay will be used to deliver the key and IV
- use the command line option `--encryption-iv-mode=fps` to specify that the IV is delivered with the key
- use the command line option `--encryption-key-uri` to specify the key URI that your DRM server will use to identify the key (this depend on how your FairPlay Key Server is implemented)

Important Note
With FairPlay, the key and IV are delivered together by the DRM. Because of that, the `--encryption-key` parameter must be a 32-byte array (64 hex characters), consisting of a 16-byte key value concatenated with a 16-byte IV value, instead of a usual 16-byte key-only value.

Example
Using FairPlay, encrypting with key=baab6d0dd153762d945d5a060abb5fcd and IV=00000000000000000000000000000000

```
mp4hls --output-single-file --encryption-mode=SAMPLE-AES --encryption-key=baab6d0dd153762d945d5a060abb5fcd00000000000000
```

Custom Key Delivery

To use a custom key delivery method, you simply use the `--encryption-key-uri` option and/or the `--encryption-key-format` option to specify a custom URI for the key, and optionally a custom key format.

Example
Encrypt, specifying a custom key URI (notice that we do not store the key in a file, since it will delivered using a custom key delivert mechanism)

```
mp4hls --encryption-key baab6d0dd153762d945d5a060abb5fcd --encryption-key-uri=myurlscheme://foobar --encryption-key-for
```

Multi-Language Audio

An HLS presentation can specify that more than one audio track (in different languages) is available to go along with the video.
The `mp4hls` tools will try to automatically detect if you have multiple audio tracks in your MP4 files with different languages. However, for efficiency reasons, it is not recommended to have multiplexed segments with multiple audio tracks. It is more efficient to convey alternate audio tracks in separate segments. To achieve this, you can pass as input to `mp4hls` audio-only MP4 files, or multiplexed MP4 files prefixed with the filter `[type=audio]`. In addition, if the language code of the tracks in the input MP4 files isn't set correctly, you can override it by prefixing the filename with the

`[+language=XX]` specifier. If you need both a filter and a language override, they get combined, separated by commas (ex: `[type=audio,+language=fr]').`
`Finally,` mp4hls `uses the language name as the` NAME `field of the alternate audio tracks (which may appear in the user interface on the client), but this can be overridden with` [+language_name=YYYYYYY]`

Examples
Main video with english audio, and alternate audio track in French

```
mp4hls video_with_english_audio.mp4 french_audio.mp4
```

Main video with english audio, and alternate audio track in French, where the language code needs to be overridden.

```
mp4hls video_with_english_audio.mp4 [+language=fr]audio.mp4
```

Main video with english audio with an overridden language name, and alternate audio track in French, where the language code needs to be overridden.

```
mp4hls [+language_name=Main]video_with_english_audio.mp4 [+language=fr]audio.mp4
```

Subtitles

`mp4dash` supports subtitles in WebVTT format. In order to include subtitles with a presentation, you need to set the subtitles input file(s) type by prefixing the filename(s) on the command line with `[+format=webvtt]`.
Since WebVTT files usually do not contain an explicit language code, it is often necessary to set the language using the `[+language=XX]` prefix.

Example
English language subtitles

```
mp4hls video_00500.mp4 [+format=webvtt,+language=en]sub_eng_webvtt.txt
```

English and French subtitles

```
mp4hls video_00500.mp4 [+format=webvtt,+language=en]sub_eng_webvtt.txt [+format=webvtt,+language=fr]sub_fre_webvtt.txt
```

Using an audio-only fallback stream

It may be useful, or sometimes even necessary, to include in the list of video variant bitrates an audio-only low bitrate stream (typically 64kbps). Since such an input may be confused with an alternate audio track, you must prefix it with `[+audio_fallback=yes]` in order for it to be recognized as such.

Example

```
mp4hls video_00500.mp4 [+audio_fallback=yes]audio_64kbps.mp4
```