Adobe® Primetime

PSDK 1.2 for iOS Programmer's Guide

Updated 11 April 2014

Contents

PSDK 1.2 for iOS Programmer's Guide

Overview

The Adobe Primetime Software Development Kit (PSDK) client for iOS is a toolkit that provides you with the means to add advanced video playback functionality to your applications. This PSDK also supports ad insertion, content protection, and analytics.

The PSDK for iOS is implemented entirely in Objective-C . It includes an API and sample code to help you create an advanced media player and integrate Primetime features into the player.

This guide assumes that you understand developing applications using Objective-C. It walks you through the steps required to create a video player in Objective-C on iOS devices using the Primetime Player SDK.

Overview of the Player SDK

The Primetime Player SDK for iOS includes a variety of features.

The following list describes some of the PSDK’s functionality:

- VOD and live/linear streaming
- Support for full-event replay
- Seamless ad insertion and tracking through VAST/VMAP
- Access to digital rights management (DRM)-related services
- Playback of HLS streams unencrypted or with Protected HTTP Live Streaming (PHLS), AES128, or
- Client tracking and QoS measurement and reporting
- Notifications that enable the PSDK and your application to communicate asynchronously about the status of videos, advertisements, and other elements, and also that log activity
- Management of the playback window, including methods that play, stop, pause, seek, and retrieve the playhead position
- Integration with Video Analytics
- Closed captioning (608, WebVTT) and alternate forms of audio for increased accessibility
- DVR capability
- Bit-rate controls and adaptive bit-rate logic
- Custom cue tags for ads
- Failover
- Debug logging
- Adjustable playback buffers
- Subscription to HLS and non-HLS tags

Considerations

Keep this information in mind while using the PSDK for iOS .

- Playback is supported only for HTTP Live Streaming (HLS) content.
- Video playback requires the native Apple AV Foundation framework. This affects how and when media resources, including closed captions and timelines, can be accessed:
  - Timeline adjustments cannot be revised after the initial setup.

For example, an advertisement cannot be removed from the timeline after it has played, so if the user seeks back in the presentation, the same ad plays again even if the normal policy would have been to remove it.

- Depending on encoder precision, the actual encoded media duration may differ from the durations recorded in the stream resource manifest.

  There is no reliable way to resynchronize between the ideal virtual timeline and actual playout timeline. Progress tracking of the stream playback for ad management and Video Analytics must use the actual playout time, so reporting and user interface behavior might not precisely track the media and advertisement content.

Requirements

The iOS player requires specific properties for content.

- Content segment duration

  The duration of any given segment must not exceed the target duration specified in the manifest file.
- Content segment key frames

  Each content segment must begin with a key frame.
- Sequence numbers

  In live/linear video, sequence numbers must match between all bit-rate renditions for main content at any given time.

Best practices

These are recommended practices for the Adobe PSDK for iOS .

- Use HLS Version 3.0 or above for program content.
- Use Apple’s mediastreamvalidator tool to validate VOD streams.

New features for 1.2

- **Video Analytics integration**

  Video Analytics is Adobe's video-tracking solution, which is available to you through the Primetime SDK.

  The Adobe Video Analytics feature encompasses two aspects of video tracking:

  - *Primetime Player Monitoring* - This is provided with the PSDK. It provides real-time metrics about the quality of streaming, including the buffer rate, error rate, average bit rate, active streams, and startup time.
  - Adobe Analytics Video Essentials - This is an add-on service. It provides video engagement metrics after the fact (non real-time), including video views, video completes, ad impressions, time spent on video, and more. For more information, contact your Adobe representative.
- **On-the-fly transcoding or creative repackaging**

  Some third-party ads (creatives) might be available only as a progressive download MP4 video, in which case they cannot be stitched into the HLS content stream. Primetime Ad Insertion and the iOS PSDK provide an option called third-party creative repackaging, which addresses this situation.

On the fly ad transcoding is supported through Adobe Ad Serving. The Adobe Ad Server account must be configured for creative repackaging on the Ad Server and then repackaging is seamlessly handled through the PSDK.

- **Full-event replay (FER)**

  Full-event replay (FER) is a VOD asset that acts, in terms of ad insertion, as a live/DVR asset. Your application can now ensure that ads are placed correctly by selecting the ad-signaling mode.

- **General playback stability**

  Improved start-up time, failover, and playback performance.

- **Pre-roll playback improvement for live streams**

  Playback of pre-roll ads is now separated from the playback of the main content for live streams. This will improve the startup time by loading the mid-roll ads and initializing DRM workflow pre-hand while the pre-roll is still playing. Seeking back in the content won't return the user to the pre-roll ad.again.

- **iOS 7 and armv7s support**
- **Ad Optimization**

  Optimized playback startup time of DVR LIVE and VOD FER streams by making a single ad server call instead of making individual requests per ad cue point.

- **Support for EVENT playlist**

  PSDK iOS build supports EVENT type HLS playlists. Fixes issues with Ad Tracking.

- **Support for ad signaling mode**

  PSDK advertising workflow can be split in 3 phases:

  1. Opportunity detection - PSDK will use stream information to detect the possible/desired locations for ads
  2. Ad resolution - PSDK will communicate with an advertisement server to retrieve the ads which need to be spliced into the content
  3. Ad placement - PSDK will load specified ads and place them on the content timeline at the specified locations

  There are two ways in which the PSDK can obtain the possible locations for ad placement:

  1. Manifest metadata/cues

     This is a common scenario for live/linear streams. The PSDK is responsible for detecting such metadata/cues, extracting the necessary information from them and communicating with an advertising server to obtain the corresponding ads.

     The resolved ads are spliced in by replacing main content at the location indicated by the metadata/cues. Otherwise the client will drop further and further than the actual live point.

  2. Advertising server map

     This is a common scenario for video-on-demand streams. Metadata information about these streams are registered into the advertising server prior to playback. The PSDK is responsible for retrieving the ad timeline from the server and corresponding ads. The resolved ads are spliced by insertions into the main content as indicated by the server map.

  By default, PSDK uses manifest cues for live and linear streams and an advertising server map for VOD streams. This works well for simple playback scenarios, but we can complete live streams using the metadata/cues presented in the manifest. For HLS, once the EXT-X-ENDLIST tag is appended to a live manifest, the stream becomes a VOD stream and there is no way to differentiate it from a normal VOD stream. The PSDK can correctly identify these streams and play them only if it is instructed to do this by the application. Use *PTAdSignalingMode* to control this behavior. The PSDK assumes that the application will obtain the actual signaling mode that should be used from external sources (vCMS, etc.).

- **DVR support for ad insertion**

  The PSDK provides you with the option to load all ads for the stream when the user begins viewing or to resolve only the ads that occur after the user's current live point.
- **LBA support for live streams**

  PSDK 1.2 enhances LBA support to include the ability to have a live stream with multiple audio tracks.

  For more information about LBA support, refer to the *PSDK for iOS API Reference*.
- **Ad signaling mode**

  The application developers must set the PTAdSignalingMode in *PTAdMetadata* to define the ad signaling mode.

  These are the types of signaling modes:

  - PTAdSignalingModeDefault
  - PTAdSignalingModeManifestCues
  - PTAdSignalingModeServerMap
- **Added volume and muted properties in iOS 7**

  The properties volume(float) and muted(BOOL) were added to PTMediaPlayer to set/get the volume of the playback for iOS 7.
- **Support for using registered custom Ad Cue markers to splice ads content**

  Client can register custom ad cue markers for the PSDK to use in addition to the default ad markers it always checks for. This can be done by using the class method setAdTags: in the *PTSDKConfig* class.
- **Support for exposing subscribed custom tags** within the HLS manifest to the application.

  Clients can register with the PSDK to be notified about a custom tag within the HLS manifest. This can be done by using the class method setSubscribedTags: in the *PTSDKConfig* class.

Create a video player

The PSDK provides the tools that you need for creating your own advanced video player application, which you can integrate with other Primetime components.

Follow these instructions to create your own Primetime player:

Set up the PTMediaPlayer

The `PTMediaPlayer` interface encapsulates the functionality and behavior of a media player object.

Set up your `PTMediaPlayer` based on the example below.

```
// 1. Fetch the URL from UI, maybe in a text field.
NSURL  *url  =  [NSURL  URLWithString:textFieldURL.text];

// 2. Create PTMetadata.
// Assume that the createMetada method prepares metadata (Described in Including advertising).
PTMetadata *metadata = [self createMetadata];

// 3. Create PTMediaPlayerItem using the PTMetadata instance created above.
```

```
PTMediaPlayerItem *item =
  [[[PTMediaPlayerItem alloc] initWithUrl:url mediaId:yourMediaID metadata:metadata]
autorelease];

// 4. Add observers to notifications dispatched from the PSDK.
// Assume that the addObservers method does that (Described in Set up notifications).
[self addObservers]

// 5. Create PTMediaPlayer using the PTMediaPlayerItem created above.
PTMediaPlayer *player = [PTMediaPlayer playerWithMediaPlayerItem:item];

// 6. Set properties on the player. For example:
player.autoPlay = YES;
player.closedCaptionDisplayEnabled = YES;
player.videoGravity = PTMediaPlayerVideoGravityResizeAspect;
player.allowsAirPlayVideo = YES;

// 7. Set the player's view property.
CGRect playerRect = self.adPlayerView.frame;
playerRect.origin = CGPointMake(0, 0);
playerRect.size = CGSizeMake(self.adPlayerView.frame.size.width,
self.adPlayerView.frame.size.height);

[player.view  setFrame:playerRect];
[player.view setAutoresizingMask:( UIViewAutoresizingFlexibleWidth |
UIViewAutoresizingFlexibleHeight  )];

// 8. Add the player's view in the current view's subview.
[self.adPlayerView  setAutoresizesSubviews:YES];
[self.adPlayerView addSubview:(UIView *)player.view];

// 9. Call play on the player.
[player play];
```

Set up notifications

The player can listen for a range of events that indicate the state of the player.

Assuming that `PTMediaPlayer` is a property of the client player, `self.player` in the following example represents the `PTMediaPlayer` instance. The following example implements the `addObservers` method shown in the PTMediaPlayer setup instructions, and includes most of the notifications:

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerStatusChange:)
name:PTMediaPlayerStatusNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerNotification:)
name:PTMediaPlayerNewNotificationEntryAddedNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerTimeChange:)
name:PTMediaPlayerTimeChangeNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerItemPlayStarted:)
name:PTMediaPlayerPlayStartedNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerItemPlayCompleted:)
name:PTMediaPlayerPlayCompletedNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerItemTimelineChanged:)
name:PTMediaPlayerTimelineChangedNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerItemMediaSelectionOptionsAvailable:)
name:PTMediaPlayerMediaSelectionOptionsAvailableNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerAdBreakStarted:)
name:PTMediaPlayerAdBreakStartedNotification object:self.player];
```

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerAdBreakCompleted:)
name:PTMediaPlayerAdBreakCompletedNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerAdPlayStarted:)
name:PTMediaPlayerAdStartedNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerAdPlayProgress:)
name:PTMediaPlayerAdProgressNotification object:self.player];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerAdPlayCompleted:)
name:PTMediaPlayerAdCompletedNotification object:self.player];
```

List of iOS Notifications

The`PTMediaPlayerNotifications` class lists the notifications that the PSDK dispatches to your player.

Notification	**Meaning**
PTMediaPlayerAdBreakCompletedNotification	An ad break ended.
PTMediaPlayerAdBreakStartedNotification	An ad break started.
PTMediaPlayerAdClickNotification	A user clicked a banner ad.
PTMediaPlayerAdCompletedNotification	An individual ad ended.
PTMediaPlayerAdProgressNotification	An ad progressed; dispatched constantly while an ad plays.
PTMediaPlayerAdStartedNotification	An individual ad started.
PTMediaPlayerItemChangedNotification	A different PTMediaPlayerItem of the PTMediaPlayer has been set.
PTMediaPlayerItemDRMMetadataChanged	DRM metadata changed.
PTMediaPlayerMediaSelectionOptionsAvailableNotification	There are new subtitles and alternate audio tracks (PTMediaSelectionOption).
PTMediaPlayerNewNotificationEntryAddedNotification	A new PTNotification has been added to the PTNotificationHistoryItem of the current PTMediaPlayerItem, that is, when a notification event is added to the notification history.
PTMediaPlayerPlayCompletedNotification	Media playback ended.
PTMediaPlayerPlayStartedNotification	Playback started.
PTMediaPlayerStatusNotification	The player status changed. Possible status values are: • PTMediaPlayerStatusCreated • PTMediaPlayerStatusInitializing • PTMediaPlayerStatusInitialized • PTMediaPlayerStatusReady • PTMediaPlayerStatusPlaying • PTMediaPlayerStatusPaused • PTMediaPlayerStatusStopped • PTMediaPlayerStatusCompleted

	• PTMediaPlayerStatusError
PTMediaPlayerTimeChangeNotification	The playback current time changed.
PTMediaPlayerTimelineChangedNotification	The current player timeline changed.
PTTimedMetadataChangedNotification	The PSDK encountered the first occurrence of a subscribed tag.

Sample Handlers For Notifications

The following code snippets illustrate some of the ways you can use notifications.

Fetch the PTAdBreak instance using PTMediaPlayerAdBreakKey:

```
 - (void) onMediaPlayerAdBreakStarted:(NSNotification *) notification {
   // Fetch the PTAdBreak instance using PTMediaPlayerAdBreakKey
   PTAdBreak *adBreak = [notification.userInfo objectForKey:PTMediaPlayerAdBreakKey];
   ...
   ...
}
```

Set subtitlesOptions and audioOptions:

```
 - (void) onMediaPlayerItemMediaSelectionOptionsAvailable:(NSNotification \*) notification {
   //SubtitlesOptions and audioOptions are set and accessible now.
   NSArray* subtitlesOptions = self.player.currentItem.subtitlesOptions;
   NSArray* audioOp tions = self.player.currentItem.audioOptions;
   ...
   ...
}
```

Fetch the PTAd instance using PTMediaPlayerAdKey:

```
 - (void) onMediaPlayerAdPlayStarted:(NSNotification \*)  notification {
   // Fetch the PTAdinstance using PTMediaPlayerAdKey
   PTAd *ad = [notification.userInfo objectForKey:PTMediaPlayerAdKey];
   ...
   ...
}
```

Configure the Player User Interface

With the PSDK, you can control the basic playback experience for live and video on demand (VOD). The PSDK does not configure the player for you; instead, it provides methods and properties on the player instance that you can use to configure the player user interface.

The following sections illustrate a variety of ways that you can configure the user interface:

Implement a Play/Pause Button

You can set up buttons that call PSDK methods to pause and play the media.

Implement a Play/Pause button using the following sample code as a guide:

```
_playPauseButton =
[[UIButton alloc] initWithFrame:CGRectMake(BUTTON_POS_X, BUTTON_POS_Y, BUTTON_SIZE_W,
BUTTON_SIZE_H)];
[_playPauseButton setImage:[UIImage imageNamed:@"play.png"] forState:UIControlStateNormal];
[_playPauseButton setImage:[UIImage imageNamed:@"pause.png"] forState:UIControlStateSelected];
```

```
[_playPauseButton addTarget:self action:@selector(playTouch:)
forControlEvents:UIControlEventTouchUpInside];
[self addSubview:_playPauseButton];

...

- (void)playTouch:(id)sender {
    if (self.player.status == PTMediaPlayerStatusPlaying) {
        _playPauseButton.selected = YES;
        [self.player pause];
    }
    else {
        _playPauseButton.selected = NO; [self.player play];
    }
}
```

Display the Duration of the Video

You can display the duration of the current active content.

Implement a video-duration display using the following sample code as a guide.

The *PTMediaPlayer* property, *seekableRange* , contains the current seekable window range:

- For VOD, this range is the entire VOD content range, with ads included.
- For live/linear, this range represents the seekable window.

```
CMTimeRange seekableRange = self.player.seekableRange;
if (CMTIMERANGE_IS_VALID(seekableRange)) {
    double start = CMTimeGetSeconds(seekableRange.start);
    double duration = CMTimeGetSeconds(seekableRange.duration);
}
```

Display the Current Time and Remaining Time

You can display the current and remaining time of the content you are showing.

Implement a display that shows the current and remaining time of the active content, using the following sample code as a guide:

```
// 1. Register for the PTMediaPlayerTimeChangeNotification
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerTimeChange:)
  name:PTMediaPlayerTimeChangeNotification object:self.player];

...

// 2. Create labels for displaying current and remaining time
_timeCurrentLabel = [[UILabel alloc] initWithFrame:CGRectMake(50.0, 16.0, 50.0, 21.0)];
_timeCurrentLabel.text = @"00:00:00";
_timeCurrentLabel.font = [UIFont boldSystemFontOfSize:12.0];
_timeCurrentLabel.numberOfLines = 1;
_timeCurrentLabel.textAlignment = UITextAlignmentCenter;
_timeCurrentLabel.backgroundColor = [UIColor clearColor];
_timeCurrentLabel.textColor =
  [UIColor colorWithRed:209.0/255.0 green:209.0/255.0 blue:209.0/255.0 alpha:1.0];
[self addSubview:_timeCurrentLabel];

_timeRemainingLabel = [[UILabel alloc] initWithFrame:CGRectMake(485.0, 16.0, 50.0, 21.0)];
_timeRemainingLabel.text = @"00:00:00";
_timeRemainingLabel.font = [UIFont boldSystemFontOfSize:12.0];
_timeRemainingLabel.numberOfLines = 1;
_timeRemainingLabel.textAlignment = UITextAlignmentCenter;
_timeRemainingLabel.backgroundColor = [UIColor clearColor];
_timeRemainingLabel.textColor =
  [UIColor colorWithRed:209.0/255.0 green:209.0/255.0 blue:209.0/255.0 alpha:1.0];
```

```
...

// 3. This method is called whenever the player time changes
(PTMediaPlayerTimeChangeNotification) - (void) onMediaPlayerTimeChange:(NSNotification
*)notification {
    //The seekable range provides the playback range of a stream
    CMTimeRange seekableRange = self.player.seekableRange;

    //Verify if the seekableRange is a valid CMTimeRange
    if (CMTIMERANGE_IS_VALID(seekableRange)) {
        double  duration = CMTimeGetSeconds(seekableRange.duration);
        double currentTime = CMTimeGetSeconds(self.player.currentItem.currentTime);
        if (CMTIME_IS_INDEFINITE(self.player.currentItem.duration)) {
            //If the duration is indefinite then the content is live.
            [_timeCurrentLabel setText:[NSString stringWithFormat:@"--:--"]];
            [_timeRemainingLabel setText:[NSString stringWithFormat:@"Live"]];
        }
        else {
            [_timeCurrentLabel setText:[self timeFormatter:currentTime]];
            [_timeRemainingLabel setText:[self timeFormatter:(duration - currentTime)]];
        }
    }
}
```

Display a Seek Scrub Bar With the Current Playback Time Position

You can display the current and remaining time of the content you are showing.

Implement a scrub bar, using the following sample code as a guide:

```
// 1. Register for the PTMediaPlayerTimeChangeNotification
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerTimeChange:)
  name:PTMediaPlayerTimeChangeNotification object:self.player];

...

_positionSlider = [[UISlider alloc] initWithFrame:CGRectMake(105.0, 14.0, 370, 24)];
[_positionSlider addTarget:self action:@selector(sliderThumbReleased:)
  forControlEvents:UIControlEventTouchUpInside];

...

// 2. Cover the event where the user moves to a different location in the stream
- (void)sliderThumbReleased:(id)sender {
    double  sliderTime = [_positionSlider  value];
    CMTimeRange seekableRange = self.player.seekableRange;
    if (CMTIMERANGE_IS_VALID(seekableRange)) {
        double start = CMTimeGetSeconds(seekableRange.start);
        double duration = CMTimeGetSeconds(seekableRange.duration);

        CMTime newTime = CMTimeMakeWithSeconds((sliderTime * duration) + start, AD_TIMESCALE);

        [self.player seekToTime:newTime];
    }
}

...

// 3. This method is called whenever the player time changes
(PTMediaPlayerTimeChangeNotification)
- (void) onMediaPlayerTimeChange:(NSNotification *)notification {
    CMTimeRange seekableRange = self.player.seekableRange;

    if (CMTIMERANGE_IS_VALID(seekableRange)) {
        double start = CMTimeGetSeconds(seekableRange.start);
        double duration = CMTimeGetSeconds(seekableRange.duration);
        double currentTime = CMTimeGetSeconds(self.player.currentItem.currentTime);

        if (duration > 0) {
            //Set the position slider value on the current playback time
            [_positionSlider setValue:((currentTime - start) / duration)];
        }
    }
}
```

Control the quality with adaptive bit rates (ABR)

The PSDK can play video assets that have multiple bit rates to provide more than one quality level.

Based on the bandwidth conditions and the quality of playback (frame rate), the video engine automatically switches the quality level to provide the best playback experience. The PSDK performs transitions between the various quality levels seamlessly and automatically. PSDK exposes the list of renditions and enables you to control some aspects of the adaptive streaming experience.

You can specify multiple profiles that have different bit rates.

To configure the adaptive bit-rate (ABR) engine, choose values for the ABR parameters.

1. Decide on the minimum and maximum bit rates.

   These define a range of bit rates from which the ABR engine can choose. The ABR engine uses profiles only from this range when downloading fragments and rendering images on screen.
2. Decide on the initial bit rate, in bits per second, to use when beginning playback.

   For preload of the first downloaded fragment, the PSDK selects the profile with the bit-rate value that is closest to (equal or less) this initial value. For the first fragment, the minimum/maximum range is ignored.
3. Assign the ABR parameters to the media player.

   ```
   @property (assign, nonatomic) int initialBitRate
   @property (assign, nonatomic) int maxBitRate
   @property (assign, nonatomic) int minBitRate
   - (id)initWithABRControlParameters:(int)initialBitRate minBitRate:(int)minBitRate
   maxBitRate:(int)maxBitRate
   ```

Monitor QoS Statistics

You can monitor QoS statistics by using an `NSTimer` (or other methods) to scan the information provided by `PTQOSProvider`.

To monitor QoS statistics with a `PTQOSProvider`:

1. Create the `PTQOSProvider` instance. For example:

   ```
   qosProvider = [[PTQOSProvider alloc]initWithPlayer:self.player];
   PTDeviceInformation *devInfo = qosProvider.deviceInformation;
   if (devInfo) {
       [consoleView logMessage:@"=== qosDeviceInfo:==\n os =%@\n model =
          %@\n id =%@\n\n", devInfo.os, devInfo.model, devInfo.id];
   }
   [NSTimer scheduledTimerWithTimeInterval:2.0 target:self
      selector:@selector(printPlaybackInfoLog) userInfo:nil repeats:YES];
   ```
2. Read the values provided by the `PTQOSProvider`. For example:

   ```
   - (void)printPlaybackInfoLog {
       PTPlaybackInformation *playbackInfo = qosProvider.playbackInformation;
   ```

```
    if (playbackInfo) {
        // For example:
        NSString *infoLog = [NSString stringWithFormat:@"observedBitrate :
                                     %f\n",playbackInfo.observedBitrate];
        [consoleView logMessage:@"====%@\n\n",infoLog];
    }
}
```

Work with MediaPlayer objects

The `PTMediaPlayer` object represents your media player and a `PTMediaPlayerItem` represents audio or video on your player.

About the MediaPlayerItem class

Successfully loading a media resource creates an instance of the `PTMediaPlayerItem` class.

The The `PTMediaPlayer` starts by resolving the media resource and continues by loading the associated manifest file and parsing it (this is the asynchronous part of the resource loading process). At the end of the resource resolving process, the `PTMediaPlayerItem` instance is produced, so the `PTMediaPlayerItem` instance is basically the resolved version of a media resource.

The PSDK provides access to the newly created `PTMediaPlayerItem` instance through `PTMediaPlayer.currentItem`. Wait for the resource to be successfully loaded before accessing the media player item.

Lifecycle and states of the MediaPlayer object

From the moment when the `PTMediaPlayer` instance is created to the moment it is terminated (reused or removed), the `PTMediaPlayer` object completes a series of transitions from one status to another.

The list of statuses is defined in `PTMediaPlayerStatus` . You can retrieve the current of the `PTMediaPlayer` object with `PTMediaPlayer.status` .

Knowing the player's status is useful because some operations are permitted only while the player is in a particular status . For example, `play` cannot be called while in PTMediaPlayerStatusCreated . It must be called after reaching the PTMediaPlayerStatusReady status .

The PTMediaPlayerStatusError status also changes what can happen next.

Here is how the basic procedure for loading a media resource inside the `PTMediaPlayer` corresponds to state transitions:

1. The initial status is PTMediaPlayerStatusCreated .
2. Your application calls `PTMediaPlayer.replaceCurrentItemWithPlayerItem` , which moves the player to PTMediaPlayerStatusInitializing .
3. The PSDK loads the resource. If successful, the status becomes PTMediaPlayerStatusInitialized .
4. Your application calls `PTMediaPlayer.prepareToPlay`.
5. The PSDK prepares the media stream and starts the ad resolving and ad insertion (if enabled).
6. When this finishes (either ads are inserted into the timeline or the ad procedure has failed), the player status becomes PTMediaPlayerStatusReady .

7. As your application plays and pauses the media, the status moves between PTMediaPlayerStatusPlaying and PTMediaPlayerStatusPaused .
8. When the player reaches the end of the stream, the status becomes PTMediaPlayerStatusCompleted .
9. When your application releases the media player, the status becomes PTMediaPlayerStatusStopped .
10. If an error occurs at any time during the process, the status becomes PTMediaPlayerStatusError .

You can use the status to provide feedback to the user on the process (for example, a spinner while waiting for the next status change) or to take the next steps in playing the media, such as waiting for the appropriate status before calling the next method.

Include advertising

The Primetime SDK for iOS allows you to request ads for your live/linear and VOD content through its Primetime Ad Decisioning interface.

Ad Decisioning works with the PSDK to identify ad opportunities, resolve ads, and insert resolved ads into your video streams.

Advertising Requirements

There are several requirements to be aware of while incorporating ads in your video content.

The following requirements apply to ad insertion:

- The HLS version of the advertising content must be the same as or earlier than the HLS version of the main content.
- Target duration and any individual segment duration of the ad must not exceed the target duration of the main content.
- Advertising content must have an audio-only stream if the main content also contains an audio-only stream.
- Advertising content must not be encrypted if the main content has subtitles streams.
- Advertising content must be multiple bit rate (MBR) if the main content is MBR.
- Ad playlists must have the same bit-rate renditions as that of the main content playlist.
- If the main content has alternate audio tracks, each ad must have at least an audio-only stream, or the ad is skipped.

Basic Ad Insertion

Basic ad insertion includes ad resolving (for VOD, live streaming, and linear streaming), along with ad tracking and ad playback.

The PSDK makes the required requests to the AdServing ad server, receives information about ads for the specified content, and splices the ads into the content.

VOD Ad Resolving and Insertion

The PSDK supports several use cases for VOD ad resolving and insertion.

In VOD, the PSDK supports the following use cases:

- Pre-roll ad insertion

  PSDK inserts ads at the beginning of the content.
- Mid-roll ad insertion

  PSDK inserts one or more ads in the middle of the content.

• Post-roll ad insertion

PSDK appends one or more ads at the end of the content.

The PSDK resolves the ads, inserts them at locations defined by the ad server, and computes the virtual timeline prior to starting playback. Once playback starts, no further changes can occur in the content. For example:

• No additional ad insertion will occur
• No inserted ads will be removed

Live and Linear Ad Resolving and Insertion

The PSDK supports several use cases for live and linear ad resolving and insertion.

The PSDK supports the following live and linear use cases:

• Pre-roll ad insertion

PSDK inserts one or more ads at the beginning of the content.

• Mid-roll ad insertion

PSDK inserts one or more ads in the middle of the content.

The PSDK resolves the ads and inserts them when a cue point is encountered in the live or linear stream. By default, the PSDK supports the following cues as valid ad markers when resolving and placing ads:

• #EXT-X-CUEPOINT
• #EXT-X-AD
• #EXT-X-CUE
• #EXT-X-CUE-OUT

These markers require the metadata field's DURATION in seconds and the cue’s unique ID. For example:

```
#EXT-X-CUE DURATION=27 ID=identiferForThisCue ...
```

You can define additional cues as described in “Subscribing to custom HLS tags".

Client Ad Tracking

The PSDK includes the Ad Serving ad tracking library.

The PSDK includes the Ad Serving ad tracking library, which automatically tracks ads for both VOD and live/linear streaming. You can also track information for video analytics.

Ad Playback

The PSDK includes the ad playback functionality, with some considerations.

For VOD and live/linear streaming, timeline adjustments cannot be revised once made, which means that an advertisement cannot be removed from the timeline after it has played. Therefore, if the user seeks back in the presentation, the same ad plays again even if the normal policy would have been to remove it.

Primetime Ad Serving Metadata

The PSDK supports resolving and inserting ads for VOD and live/linear streams.

The PSDK includes the AdServing library, which resolves and inserts ads for both VOD and live/linear video streams. For detailed information about the Primetime Ad Serving platform, refer to the Ad Serving documentation:

Metadata Prerequisites For Ad Insertion

You need to provide some metadata when you include advertising in your content.

To include advertising in your video content, you need to provide the following information:

- A `mediaID` (identifies the specific content to play)
- Your `zoneID` (identifies your company or Web site)
- Your Auditude ad server domain (specifies the domain of your assigned ad server)
- Other targeting parameters

Set Up Primetime Ad Serving Metadata

Your application must provide the PSDK with the required *`PTAuditudeMetadata`* information for connecting to the Ad Serving ad server.

To set up the Ad Serving metadata:

1. Create an instance of `PTAuditudeMetadata` and set its properties:

```
PTAuditudeMetadata *adMetadata = [[PTAuditudeMetadata alloc] init];
adMetadata.zoneId = @"INSERT_HERE_ZONE_ID";
adMetadata.domain = @"INSERT_HERE_DOMAIN";
```

2. Set the `PTAuditudeMetadata` instance as metadata for the current `PTMediaPlayerItem` metadata using the key `PTAdResolvingMetadataKey`:

```
// Metadata is an instance of PTMetadata that is used to create the PTMediaPlayerItem
[metadata setMetadata:adMetadata forKey:PTAdResolvingMetadataKey];
[adMetadata release];
```

```
PTMetadata *metadata = [self createMetadata];
PTMediaPlayerItem *item =
  [[[PTMediaPlayerItem alloc] initWithUrl:url mediaId:yourMediaID metadata:metadata]
autorelease];

- (PTMetadata *) createMetadata {
    PTMetadata* metadata = [[[PTMetadata alloc] init] autorelease];

    PTAuditudeMetadata *adMetadata = [[[PTAuditudeMetadata alloc] init] autorelease];
    adMetadata.zoneId = yourZoneID;
    adMetadata.domain = yourAdServerDomain;

    [metadata setMetadata:adMetadata forKey:PTAdResolvingMetadataKey];

    return metadata;
}
```

Enable ads in full-event replay

Full-event replay (FER) is a VOD asset that acts, in terms of ad insertion, as a live/DVR asset, so your application must take steps to ensure that ads are placed correctly.

For live content, the PSDK uses the metadata/cues presented in the manifest to determine where to place ads. However, sometimes live/linear content looks like VOD content, for example, when live content completes, an EXT-X-ENDLIST tag is appended to the live manifest. For HLS, the presence of the EXT-X-ENDLIST tag means that the stream is a VOD stream, and there is no way for the PSDK to automatically differentiate it from a normal VOD stream so that it can insert ads correctly.

Therefore, your application must notify the PSDK of this situation by specifying the `AdSignalingMode`.

For a FER stream, you do not want the Ad Decisioning server to provide the list of ad breaks that need to be inserted on the timeline prior to beginning playback, as would be usual for VOD. Instead, by specifying a different signaling mode, the PSDK reads all the cue points from the FER manifest and goes to the ad server for each cue point to request an ad break (similar to live/DVR).

Besides each request associated with a cue point, the PSDK makes an additional ad request for pre-roll ads.

1. Obtain from an external source (vCMS, etc.) the signaling mode that should be used.
2. Create advertising-related metadata as usual.
3. Specify the `PTAdSignalingMode` using `AdvertisingMetadata.setSignalingMode` if the default behavior must be overridden.

   Valid values are PTAdSignalingModeDefault, PTAdSignalingModeManifestCues, PTAdSignalingModeServerMap.

   You must set the ad signaling mode before calling `prepareToPlay`. After the PSDK has started resolving and placing ads on the timeline, any change to the ad signaling mode is ignored. The recommended way is to set it when creating the advertising metadata for the resource (when creating the advertising metadata for the resource).
4. Continue to playback as usual.

```
PTMetadata *metadata = [[[PTMetadata alloc] init] autorelease];
PTAuditudeMetadata *adMetadata = [[[PTAuditudeMetadata alloc] init] autorelease];
adMetadata.zoneId = your-auditude-zone-id;
adMetadata.domain = @"your-auditude-domain";
//adMetadata.enableDVRAds = YES; // FOR LIVE DVR case
//adMetadata.signalingMode = PTAdSignalingModeManifestCues;
// FOR VOD FER case
NSMutableDictionary *targetingParameters = [[[NSMutableDictionary alloc] init] autorelease];
[targetingParameters setValue:@"ipad" forKey:@"device"];
[targetingParameters setValue:@"preroll" forKey:@"AD_OPPORTUNITY_ID"];
adMetadata.targetingParameters = targetingParameters;
NSMutableDictionary *customParameters = [[[NSMutableDictionary alloc] init] autorelease];
[customParameters setValue:@"your-media-id" forKey:@"NW_ID"];
[customParameters setValue:@"preroll" forKey:@"AD_OPPORTUNITY_ID"];
adMetadata.customParameters = customParameters;
[metadata setMetadata:adMetadata forKey:PTAdResolvingMetadataKey];
```

Ad signaling mode

The ad signaling mode specifies from where the video stream should get advertising information.

Valid values are PTAdSignalingModeDefault, PTAdSignalingModeManifestCues, and PTAdSignalingModeServerMap.

The following table describes the effect of AdSignalingMode values for various HLS stream types.

	Default	**Manifest cues**	**Ad server map**
Video on Demand	• Uses server map for placement detection • Ads are inserted	• Uses in-stream cues for placement detection • Pre-roll ads are inserted in the main stream • Mid-rolls ads replace main stream	• Uses server map for placement detection • Ads are inserted
Live/linear	• Uses manifest cues for placement detection	• Uses in-stream cues for placement detection	Not supported

	Default	Manifest cues	Ad server map
	• Ads replace main stream	• Ads replace main stream	

Companion Banner Ads

The PSDK supports companion banner ads.

The PSDK supports companion banner ads, which are ads that display along with a linear ad and often remain on the page after the linear ad ends. These banners can be specified by an HTML snippet or by a URL to an iFrame page. The PSDK provides companion banner support through the *`PTAdBannerView`* class.

The PSDK provides a list of companion banner ads associated with a linear ad through the `PTMediaPlayerAdPlayStartedNotification` notification.

***Note:** Your application is responsible for displaying the companion banners that are provided with the notification.*

Display Banner Ads

The PSDK provides support for displaying banner ads.

To display banner ads:

1. Create a new `PTAdBannerView` instance for each banner. For example:

```
- (void) onMediaPlayerAdPlayStarted:(NSNotification *) notification {
    _currentAd  = [notification.userInfo  objectForKey:PTMediaPlayerAdKey];
    if (_currentAd != nil) {
        [self removeAllBanners]; // remove any existing PTAdBannerView views

        // banners
        if (_currentAd.companionAssets && _currentAd.companionAssets.count > 0) {
            PTAdAsset *bannerAsset = [_currentAd.companionAssets objectAtIndex:0];

           PTAdBannerView *bannerView = [[PTAdBannerView alloc] initWithAsset:bannerAsset];

            bannerView.player = self.player;
            bannerView.delegate = self;

            bannerView.frame = CGRectMake(0.0, 0.0, bannerAsset.width, bannerAsset.height);

            [_adBannerView.bannerView addSubview:bannerView];
        }
    }
}
```

2. Add a listener for the `PTMediaPlayerAdClickNotification` notification.

   When an end user clicks a banner, the PSDK dispatches this notification, including information about the destination for the click.

3. In your listener, use the information to display the appropriate destination URL, either by opening an internal Web page or by redirecting to an external browser, as shown:

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerAdClick:)
  name:PTMediaPlayerAdClickNotification object:self.player];

- (void) onMediaPlayerAdClick:(NSNotification *) notification {
    NSString *url = [notification.userInfo objectForKey:PTMediaPlayerAdClickURLKey];
    if (url != nil) {
        [self  openBrowser:[NSURL  URLWithString:url]];
    }
}
```

Clickable ads

The PSDK provides you with information so that you can act upon click-through ads. As you create your player UI, you are responsible for deciding how to respond when a user clicks on a clickable ad.

For the iOS PSDK, only linear ads can be clickable.

Respond to clicks on ads

When a user clicks on an ad or a related button, your application is responsible for responding. The PSDK provides you with information about the destination URL for the click.

1. Set up an event listener for the PSDK to provide you with click-through information.
   a) Add an observer for `PTMediaPlayerAdClickNotification`.
   This enables your application to receive PSDK events that provide information about the destination URL for a click on an ad.
2. Monitor user interactions on clickable ads.
3. When the user touches or clicks the ad or button, notify the PSDK.
   a) To do this, use `[_player notifyClick:_currentAd.primaryAsset];` .
4. Listen for the `PTMediaPlayerAdClickNotification` event from the PSDK.
5. Retrieve the click-through URL and related information.
   a) Use the `PTMediaPlayerAdClickURLKey` object.
6. Pause the video.
7. Display the ad click-through URL and any related information.
   For example, you could display it in one of the following ways:
   - Open the click-through URL in a browser within your application.

     On desktop platforms, the video ad playback area is typically used for invoking click-through URLs upon user clicks.
   - Redirect the user to their external mobile web browser.

     On mobile devices, the video ad playback area is used for other functions, such as hiding and showing controls, pausing playback, expanding to full screen, and so on. Therefore, on mobile devices, a separate view, such as a sponsor button, is usually presented to the user as a means to launch the click-through URL.
8. Close the browser window in which the click-through information is displayed and resume playing the video.

For example:

```
// Listening for click notification
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(onMediaPlayerAdClick:)

 name:PTMediaPlayerAdClickNotification object:self.player];
- (void) onMediaPlayerAdClick:(NSNotification *) notification {
   NSString *url = [notification.userInfo objectForKey:PTMediaPlayerAdClickURLKey];
   if (url != nil) {
       [self openBrowser:[NSURL URLWithString:url]];
   }
}
```

Repackage 3rd Party Ads

Some third-party ads cannot be stitched into the HLS content stream (due to incompatible video formats). These can be repackaged with Primetime Ad Insertion.

Some third-party ads might be available only as, for example, a progressive download MP4 video, in which case they cannot be stitched into the HLS content stream. Primetime Ad Insertion and the iOS PSDK provide an option called third-party creative repackaging, which addresses this situation.

In some situations you might need to incorporate ads (creatives) that are transcoded and served by a third party into your ad insertion workflow. For example, these could be ads that are served by an agency ad server, by your inventory partner, or by an ad network.

If a requested ad is available only as an MP4, the PSDK skips that ad and issues a request to the Primetime Ad Insertion back end to repackage the ad as HLS so that a compatible version will be available the next time that the ad is encountered. The back end generates multiple-bit-rate HLS renditions of the ad and stores these on the Primetime CDN. Then, the next time the PSDK receives an ad response that points to that ad, the PSDK uses the HLS version from the Primetime CDN.

To enable this optional feature, contact your Adobe representative.

Ad Loading For a DVR Window

You can decide whether to resolve only the ads that occur after the user’s current live point, or to also resolve ads that occur before the current live point.

When a user begins viewing at the beginning of a DVR stream, the PSDK resolves all ads for the stream at that time. If the user begins viewing at some point past the beginning of the stream, you can decide whether to resolve only the ads that occur after the user’s current live point, or to also resolve ads that occur before the current live point. The first option is faster but prevents you from being able to play earlier ads if the user seeks backwards.

Control Ad Loading For a DVR Window

To control ad loading for a DVR window:

Set the `PTAdMetadata.enableDVRAds` property to YES to load all ads for the entire stream:
NO is the default, which loads ads only from the current live point forward.

For example:

```
PTMetadata *metadata = [[[PTMetadata alloc] init] autorelease];

PTAuditudeMetadata *adMetadata = [[[PTAuditudeMetadata alloc] init] autorelease];
adMetadata.zoneId = <ZoneId>;
adMetadata.domain = <Domain>;

// Enable DVR Ads by setting to YES the enableDVRAds property on PTAdMetadata
// (PTAuditudeMetadata is a subclass of PTAdMetadata)
adMetadata.enableDVRAds = YES;

[metadata setMetadata:adMetadata forKey:PTAdResolvingMetadataKey];

//Create PTMediaPlayerItem with the previously prepared metadata
playerItem = [[PTMediaPlayerItem alloc] initWithUrl:url mediaId:yourMediaID metadata:metadata];
```

Custom HLS Tags

You can subscribe with the PSDK to be notified about custom tags within HLS manifests.

For VOD and live/linear streaming, your application can subscribe with the PSDK to be notified about custom tags within HLS manifests. Your application could use custom tags for special handling, such as managing ads for blackout events.

Subscribing to Custom HLS Tags

To subscribe with the PSDK to be notified about custom tags within HLS manifests:

Pass an array that contains the custom tags to `setSubscribedTags`.

For example:

```
NSArray *customHLSTags = [NSArray
arrayWithObjects:@"#EXT-OATCLS-SCTE35",@"#EXT_CUSTOM_TAG2",nil];
[PSDKConfig  setSubscribedTags:customHLSTags];
```

Receiving Custom HLS Tag Notifications

The PSDK issues a `PTTimedMetadataChangedNotification` notification and includes this new object in the notification's `userInfo` dictionary.

When the PSDK encounters the first occurrence of a subscribed tag in the `m3u8` playlist, it creates a new `PTTimedMetadata` object for this tag. The `PTTimedMetadata` is added to the `timedMetadataCollection` in `PTMediaPlayerItem` and the PSDK issues a `PTTimedMetadataChangedNotification` notification and includes this new object in the notification's `userInfo` dictionary.

To fetch `PTTimedMetadata` information, either:

1. Set your application to add itself as a listener to this notification and fetch the object using `PTTimedMetadataKey`.

   For example:

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onTimedMetadataChanged:)
  name:PTTimedMetadataChangedNotification object:self.player.currentItem];

- (void) onTimedMetadataChanged:(NSNotification *) notification {
    NSDictionary *timedMetadataUserInfo = [[NSDictionary alloc]initWithDictionary:
notification.userInfo];
    PTTimedMetadata *newTimedMetadata = [timedMetadataUserInfo objectForKey:
PTTimedMetadataKey];
}
```

   or;

2. Access the `timedMetadataCollection` property of `PTMediaPlayerItem`, which consists of all the `PTTimedMetadata` objects that have been notified so far.

Custom HLS Relevant API

The API elements you use to implement custom HLS tag tracking includes the PTSDKConfig class, and the PTTimedMetadata class.

- *PTSDKConfig class*
- *PTTimedMetadata class*

Subtitles and Closed Captioning

Subtitle streams run in parallel with the main content. Closed captions are part of the data packets of the MPEG-2 video streams inside of the video transmission stream.

Requirements For Subtitles

- Timestamp - The `X-TIMESTAMP-MAP` value, specified in the header section of the `WebVTT` file, should match the video timestamp.
- Supported on iOS 6.1 and later.

About Subtitles

Subtitle streams run in parallel with the main content. At any given time, the `PTMediaPlayer` plays main content and ads, where main content could be live/linear or VOD, and ads could be pre-roll, mid-roll, or post-roll.

Exposing Subtitles

The PSDK notifies the player client about the availability of internal AVAsset’s `availableMediaCharacteristicsWithMediaSelectionOptions` through the `PTMediaPlayerMediaSelectionOptionsAvailableNotification` notification. The available subtitles can then be accessed through the `PTMediaPlayerItem` property, `subtitlesOptions`.

To expose subtitles:

1. The client registers itself as a listener for the `PTMediaPlayerMediaSelectionOptionsAvailableNotification` notification:

   ```
   [[NSNotificationCenter defaultCenter]
     addObserver:self selector:@selector(onMediaPlayerItemMediaSelectionOptionsAvailable:)
     name:PTMediaPlayerMediaSelectionOptionsAvailableNotification object:self.player];
   ```

2. When the client receives this notification, it implies that `PTMediaPlayerItem` has the available subtitles ready. The `onMediaPlayerItemMediaSelectionOptionsAvailable` method needs to be similar to this:

   ```
   - (void) onMediaPlayerItemMediaSelectionOptionsAvailable:(NSNotification *) notification {
       NSArray* subtitlesOptions = self.player.currentItem.subtitlesOptions;
       NSArray* audioOptions = self.player.currentItem.audioOptions;
   }
   ```

   See "Implementing Alternate Audio" for information about alternate audio tracks.

About Closed Captions

Closed captions are part of the data packets of the MPEG-2 video streams inside of the video transmission stream.

Closed captioning is supported to the extent provided by the AV Foundation framework.

Closed captioning allows people with hearing disabilities to have access to video programming by displaying the audio portion of the video as text on the screen. Closed captioning differs from subtitles in that subtitles are typically not in the same language as the audio and subtitles do not typically include information about background sounds such as the sound of a door slamming or music.

Exposing Closed Captions

To expose closed captions:

1. Turn on closed captioning in the iOS Accessability settings.
2. Set the `closedCaptionDisplayEnabled` property on the `PTMediaPlayer` object.

Set up alternate audio

Alternate (late-binding) audio is the support of multiple language tracks for HTTP video streams (live/linear and VOD) without having to duplicate and repackage the video for each audio track. Late binding of an audio track allows you to easily provide multiple language tracks for a video asset at any time before or after the asset’s initial packaging.

Alternate audio tracks in the playlist

The playlist for a video can specify an unlimited number of alternative audio tracks for the main video content. With the PSDK, you can use these alternative tracks. For example, you might want to add different languages to your video content or allow the user to switch between different tracks on their device while the content is playing. This is known as late-binding audio.

Late-binding audio is the support of multiple language tracks for HTTP video streams (live/linear and VOD) without having to modify, duplicate, or repackage the video for each audio track. Late binding of an audio track allows you to easily provide multiple language tracks for a video asset at any time before or after the asset’s initial packaging.

In order for the alternate audio to be mixed with the video track of the main media, the timestamps of the alternate track must match the timestamps of the audio in the main track.

The following requirements apply if you use alternate audio tracks and incorporate advertising:

• If the main content has alternate audio tracks, ads must at least have an audio-only stream.

• Each segment duration of an ad's audio-only stream must be equal to the segment duration of an ad's video stream.

The main audio track is included in the audio tracks collection with the label “default”. Metadata for the alternate audio streams is included in the playlist in the #EXT-X-MEDIA tags with TYPE=AUDIO.

For example, an M3U8 manifest that specifies multiple alternate audio streams might look like this:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="bipbop_audio",LANGUAGE="eng",NAME="BipBop Audio 1",
 AUTOSELECT=YES,DEFAULT=YES
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="bipbop_audio",LANGUAGE="eng",NAME="BipBop Audio 2",
 AUTOSELECT=NO,DEFAULT=NO,URI="alternate_audio_aac/prog_index.m3u8"
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="English",AUTOSELECT=YES,FORCED=NO,
 LANGUAGE="eng",URI="subtitles/eng/prog_index.m3u8"
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="English (Forced)",DEFAULT=YES,
 AUTOSELECT=YES,FORCED=YES,LANGUAGE="eng",URI="subtitles/eng_forced/prog_index.m3u8"
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="Français",AUTOSELECT=YES,FORCED=NO,
 LANGUAGE="fra",URI="subtitles/fra/prog_index.m3u8"
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=263851,CODECS="mp4a.40.2, avc1.4d400d",
 RESOLUTION=416x234,AUDIO="bipbop_audio",SUBTITLES="subs"
gear1/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=577610,CODECS="mp4a.40.2, avc1.4d401e",
 RESOLUTION=640x360,AUDIO="bipbop_audio",SUBTITLES="subs"
gear2/prog_index.m3u8
...
```

Access alternate audio tracks

The PSDK for iOS supports late-binding audio. The PSDK uses PTMediaPlayer to play a video specified in an M3U8 HLS playlist, which can contain several alternate audio streams.

Access the available audio tracks using methods and properties in the PTMediaPlayerItem class.

You can present the available tracks to the user through the player user interface. Refer to the PSDK demo application , which includes these operations in the scrub-bar controls.

Use Video Analytics in a PSDK-based Player

Video Analytics is Adobe's video-tracking solution, which is available to you through the Primetime SDK.

The Adobe Video Analytics feature encompasses two aspects of video tracking:

- Primetime Player Monitoring - This is provided with the PSDK. It provides real-time metrics about the quality of streaming, including the buffer rate, error rate, average bit rate, active streams, and startup time.
- Adobe Analytics Video Essentials - This is an add-on service. It provides video engagement metrics after the fact (nonrealtime), including video views, video completes, ad impressions, time spent on video, and more. For more information, contact your Adobe representative.

The following summarizes what you need to do to activate real-time video tracking in your PSDK-based player.

1. Initialize and configure the Video Analytics tracking components:
   - **AppMeasurement library** - Contains the low-level data collection core logic. This is where the video heartbeat data is accumulated and sent over the network.
   - **Video heartbeat library** - Contains the video-heartbeat data collection core logic. The video heartbeat library uses the AppMeasurement module in the sense that it needs access to a subset of its APIs, and delegates some work to it.

   Note: *You do not interact directly with the video heartbeat code, but instead use the PSDK API to configure and enable the video heartbeat real-time video tracking capabilities.*

   Important - The PSDK's built-in video tracking capability depends on a properly configured AppMeasurement instance. The PSDK tracking elements assume that the AppMeasurement library is already instantiated and configured before undertaking the configuration and activation of real-time video tracking. The PSDK delegates certain operations to the AppMeasurement library, thus the video tracking capabilities of the PSDK depend on the existence of a fully functional, properly configured instance of the AppMeasurement library.
2. Set up Video Analytics reporting on the server side, using Adobe Analytics Admin Tools and accessing Primetime Player Monitoring.
3. Access the Video Analytics reports on the real-time dashboard.

Initialize and configure Video Analytics

Configure Video Analytics real-time video tracking (video heartbeat) in a PSDK-based player.

The following items are required to activate real-time video tracking (video heartbeat) in a PSDK-based player for iOS :

- **Adobe PSDK for iOS**

- **Configuration / Initialization Information** - Your Adobe representative provides your specific video-tracking account information:
  - **ADBMobileConfig.json** - *It is crucial that the name and the path of this configuration file remain unchanged. This JSON config file* ***must*** *be named `ADBMobileConfig.json`. The path to this file* ***must*** *be `<source root>/AdobeMobile` .*
  - **AppMeasurement tracking server endpoint -** The URL of the Adobe Analytics (formerly SiteCatalyst) back-end collection end-point.
  - **Video Analytics tracking server endpoint -** The URL of the Video Analytics back-end collection end-point. This is where all video heartbeat tracking calls are sent.
  - **Account name** - Also known as the Report Suite ID (RSID).
  - **Job ID** - The processing job identifier. This indicates to the back-end endpoint which kind of processing to apply for the video-tracking calls.
  - **Publisher -** The name of the content publisher (the brand or television channel using the video tracking capabilities).

To configure real-time video tracking in your PSDK-based player:

1. Configure load-time options in the `ADBMobileConfig.json` resource file:

   ```
   {
       "version" : "1.1",
       "analytics" : {
           "rsids" : "YOUR_ADOBE_ANALYTICS_RSID",
           "server" : "URL_OF_THE_ADOBE_ANALYTICS_TRACKING_SERVER (provided by Adobe)",
           "charset" : "UTF-8",
           "ssl" : false,
           "offlineEnabled" : false,
           "lifecycleTimeout" : 5,
           "batchLimit" : 50,
           "privacyDefault" : "optedin",
           "poi" : []
       },
       "target" : {
           "clientCode" : "",
           "timeout" : 5
       },
       "audienceManager" : {
           "server" : ""
       }
   }
   ```

   Your load-time options are specified in this JSON-formatted configuration file. This file is bundled as a resource with the PSDK. The Primetime Player reads these values only at load time; they remain constant while your application runs. Thus the configuration steps are as follows:

   1. Confirm that the JSON config file (`ADBMobileConfig.json`) contains the appropriate values (supplied by Adobe).
   2. Confirm that this file is located in the `AdobeMobile` folder. This folder must be located in the root of your application source tree.
   3. Compile and build your application.
   4. Deploy and run the bundled application.

   For more information on these AppMeasurement settings, see *Measuring Video in Adobe Analytics*.

2. Configure and activate Video Analytics and the PSDK's video heartbeat tracking.

   Video Analytics configuration follows the usual pattern of setting a specific metadata object on the `PTMediaPlayerItem` instance that is about to be sent to the player for playback.

Sample Video Analytics configuration:

```
// Instantiate VideoAnalytics tracking metadata.
PTVideoAnalyticsTrackingMetadata *trackingMetadata =
    [[[PTVideoAnalyticsTrackingMetadata alloc]
       initWithTrackingServer:@"THE_URL_OF_THE_VIDEO_HEARTBEATS_SERVER (provided by Adobe)"
                         jobId:@"YOUR_JOB_ID"
                     publisher:@"YOUR_PUBLISHER_NAME"] autorelease];

trackingMetadata.playerName = @"the name of your player app";

// Set this to true to activate the debug tracing.
trackingMetadata.debugLogging = YES;

// Set this to true to log the URLs of the output HTTP calls.
trackingMetadata.debugTracking = YES;

// Setting this to YES activates the "quiet" mode.
trackingMetadata.trackLocal = YES;

// Explicitly activate the video heartbeat functionality.
trackingMetadata.enableHeartbeat = YES;

// Set the metadata on the ROOT metadata.
PTMetadata *root = [[[PTMetadata alloc] init] autorelease];
[root setMetadata: trackingMetadata forKey: PTVideoAnalyticsTrackingMetadataKey];

// Attach the ROOT metadata on the media item.
PTMediaPlayerItem *mediaItem = [[[PTMediaPlayerItem alloc] initWithUrl:nsurl
mediaId:@"video-id" metadata:metadata] autorelease];
// Create a new Video Analytics Tracker instance using the current media player instance to
 start tracking video analytics data
self.videoAnalyticsTracker = [[[PTVideoAnalyticsTracker alloc] initWithMediaPlayer:self.player]
 autorelease];
```

You can set mandatory and optional configuration parameters on the VideoAnalytics metadata object:

- **Mandatory configuration parameters.** (Provided by Adobe) - Provide these as input arguments to the constructor for the `PTVideoAnalyticsTrackingMetadata` class:
  - URL of the Video Analytics tracking endpoint
  - Job ID
  - Publisher Name
- **Optional configuration parameters.** These are publicly accessible instance variables that you can set on the `PTVideoAnalyticsTrackingMetadata` class:
  - `enableHeartbeat` - Explicitly activate heartbeat tracking (`trackingMetadata.enableHeartbeat = YES;`) to activate the video heartbeat calls used for real-time analytics and Video Essentials.

    Important: *To send heartbeat calls over the network, you* ***must explicitly*** *enable the video heartbeat feature. If you do not, no video heartbeat calls will be sent over the network; the legacy Milestone tracking provided by the AppMeasurement library will be used instead.*

  - `debugLogging` - Set this flag to true to activate tracing messaging. Note that setting this flag to `true` may impact performance. This logging might be useful during development and debugging efforts, but you must set this flag to `false` for the production version of your player application. Logging is disabled (`false`) by default.
  - `trackLocal` - Set this flag to `true` to activate quiet mode. In this mode, no network calls are sent over the network. You can use this mode in conjunction with setting the `debugTracking` flag to `true`. In this case, you can see the HTTP calls in the trace console without actually sending anything over the network.

- `debugTracking` - Set this flag to `true` to track URLs in HTTP calls.

Set up Video Analytics reporting on the server side

Adobe Analytics Video Essentials requires setup on the server side.

If you are using only the built-in Primetime Player Monitoring aspect of Video Analytics, you can skip this topic. Reporting setup is necessary only if you are using Adobe Analytics Video Essentials (which provides video engagement metrics). Your Adobe representative will handle most aspects of the server-side setup for Adobe Analytics reporting, but you can also see detailed documentation on the process here: *Analytics Help and Reference - Report Suite Manager*.

To complete your server-side setup using Analytics setup:

1. Enable conversion level for RSID:
   a) Access **Admin Tools.**
   b) Select **Report Suites.**
   c) Select the RSID to set up.
   d) Select **Edit Settings** > **General** > **General Account Settings.**
   e) Choose **Enabled, no Shopping Cart** in the **Conversion Level** combo box.
   f) Click **Save**.
2. Enable video tracking:
   a) Access **Admin Tools.**
   b) Select **Report Suites.**
   c) Select the RSID to set up.
   d) Select **Edit Settings** > **Video Management** > **Video Reporting.**
   e) Click **Yes, start tracking**.

Access Video Analytics reports

Access Video Analytics reports on Adobe Analytics and on Primetime Player Monitoring.

Video Analytics reports are routed to two different reporting platforms:

- Adobe Analytics
- Primetime Player Monitoring

1. Access Adobe Analytics:
   a) Select the video-tracking enabled RSID.
   b) Select Video > Video Engagement > Video Overview.

      This displays the overview report.
   c) Select a video clip.

      This displays the minute-level granularity drop-off report.

   For more information on Adobe Analytics setup, see *Adobe Analytics Documentation Home*.
2. Access Primetime Player Monitoring:
   a) Navigate to `http://rtd.adobeprimetime.com`.
   b) Log in with your credentials. (Your Adobe representative will create the account for you if you do not have one.)

An overview report displays a list of all running videos.

c) Select a video from the list.

The real-time report for the selected video is displayed.

To view examples of video reports, see *Video Reports.*

Content security using DRM

You can use the features of the Primetime digital rights management (DRM) system to provide secure access to your video content.

Primetime DRM provides a scalable, efficient workflow for digital rights management to help you deliver and protect your premium video content. You protect and manage the rights to your video content by creating licenses for each digital media file. The PSDK gives you the ability to implement and deploy content protection.

The PSDK supports AAXS integration as custom DRM workflows. This means that your application must implement the DRM authentication workflows before playing the stream using the Flash DRMManager. To enable this, the MediaPlayer provides you with the DRM manager for authentication.

Refer to the Adobe Access Help Resource Center for complete DRM documentation.

DRM interface overview

The key element of the digital rights management (DRM) system is the DRM manager.

• A reference in the PTMediaPlayer to the DRM manager object that implements the DRM subsystem:

```
@property (readonly, nonatomic) DRMManager *drmManager
```

The PSDK issues a `PTMediaPlayerItemDRMMetadataChanged` notification when DRM metadata changes.

If the DRM-protected stream is multiple bit-rate (MBR) encoded, the DRM metadata used for the variant playlist should be the same as the one used in all the bit-rate streams.

Use the notification system

The notification portion of the PSDK library allows you to create a logging and debugging system that can be useful for diagnostic and validation purposes.

`PTNotification` notifications provide information about *player status* as information, warnings, or errors. Your application can retrieve data that describes what caused the error or warning. Errors that stop the playback of the video also cause a change of the status of the player.

Note:

The PSDK also uses notification to refer to `NSNotifications` (`PTMediaPlayer` notifications) event notifications, which the PSDK dispatches to provide information about player activity. You implement event listeners to capture and respond to those. Many event notifications also cause `PTNotification` status notificatons.

The PSDK issues `PTMediaPlayerNewNotificationItemEntryNotification` when a `PTNotification` is issued.

Notification content

`PTNotification` notifications provide information related to the player's status.

The PSDK provides a chronologically sorted list of `PTNotification` notifications. Each notification contains the following:

- Time stamp
- Diagnostic metadata, including a numeric code, a name for the notification, metadata keys, and related inner notifications

The diagnostic metadata consists of the following elements:

Element	Description
type	Describes the notification event type. Depending on the platform, this property is an enumerated type with possible values of INFO, WARN, and ERROR. This is the first, and highest-level, classification criterion for the notification events.
code	The numerical representation assigned to the notification event. • Error notification events, from 100000 to 199999 • Warning notification events, from 200000 to 299999 • Information notification events, from 300000 to 399999
name	A string containing a human-readable description of the notification event, such as PLAYBACK_START.
metadata	Metadata containing relevant information about the notification stored as key/value pairs. For example, a key named `URL` would provide a URL related to the notification, such as an invalid URL that caused an error.
innerNotification	A reference to another `PTNotification` object that directly impacted this notification. An example might be a notification about an ad-insertion failure that directly corresponds to a time-line insertion conflict.

You can store this information locally for later analysis or send it to a remote server for logging and graphical representation.

Notification setup

The PSDK sets up the player for basic notifications, while you must perform the same setup for your own custom notifications.

There are two implementations for `PTNotification`:

- One for listening
- One for adding custom notifications to `PTNotificationHistory`

For listening to notifications, the PSDK instantiates the `PTNotification` class and attaches it to an instance of the `PTMediaPlayerItem`, which is attached to a PTMediaPlayer instance. There is only one `PTNotificationHistory` instance per `PTMediaPlayer`.

If you are adding customizations, your application, rather than the PSDK, must perform those set-up steps.

Listening To Notifications

There are two ways of listening to the `PTNotification` notification of the `PTMediaPlayer`:

1. Manually check the `PTNotificationHistory` of the `PTMediaPlayerItem` with a timer and check the differences:

```
//Access to the PTMediaPlayerItem
PTMediaPlayerItem *item = self.player.currentItem;
PTNotificationHistory *notificationHistory = item.notificationHistory;

//Get the list of notification events from the notification History
NSArray *notifications = notificationHistory.notificationItems;
```

2. Use the posted *NSNotification* of the `PTMediaPlayerPTMediaPlayerNewNotificationEntryAddedNotification`. Register to the `NSNotification` using the instance of the `PTMediaPlayer` from which you want to get notifications:

```
//Register to the NSNotification

[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMediaPlayerNotification:)
  name:PTMediaPlayerNewNotificationEntryAddedNotification object:self.player];
```

Implementing Notification Callbacks

Implement the notification callback by getting the `PTNotification` from the `NSNotification` user info, and reading its values by using the key `PTMediaPlayerNotificationKey`:

```
- (void) onMediaPlayerNotification:(NSNotification *) nsnotification {
    PTNotification *notification = [nsnotification.userInfo
objectForKey:PTMediaPlayerNotificationKey];
    NSLog(@"Notification: %@", notification);
}
```

Adding Custom Notifications

To add a custom notification:

Create a new `PTNotification` and add it to the `PTNotificationHistory` by using the current `PTMediaPlayerItem`:

```
//Access to the PTMediaPlayerItem
PTMediaPlayerItem *item = self.player.currentItem;
PTNotificationHistory *notificationHistory = item.notificationHistory;

//Create notification
PTNotification* notification = [[PTNotification notificationWithType:PTNotificationTypeWarning
 code:99999 description:@"Custom notification description"];

//Add notification
[notificationHistory addNotification:notification];
```

Customized Logging

You can implement your own logging system.

In addition to logging using predefined notifications (described in "*The Notification System*"), you can implement your own logging system that uses your own log messages and messages generated by the PSDK. You can use these logs to assist with troubleshooting your player applications and to provide better understanding of the playback and advertising workflow.

Customized logging uses a shared singleton instance of the `PSDKPTLogFactory`, which provides a mechanism for logging messages to multiple loggers. You define and add (register) one or more loggers to the `PTLogFactory`. This allows you to define multiple loggers with custom implementations, such as a console logger, web logger, or console history logger.

The PSDK generates log messages for many of its activities, which the `PTLogFactory` forwards to all registered loggers. In addition, your application can generate custom log messages, which are also forwarded to all registered loggers. Each logger can then filter the messages in any way that you choose and take appropriate action.

There are two implementations for `PTLogFactory`. One is for listening to logs and the second one is for adding logs to a `PTLogFactory`.

Listening to Logs

To register for listening to logs:

1. Implement a custom class that follows the protocol `PTLogger`:

   ```
   @implementation PTConsoleLogger

   + (PTConsoleLogger *) consoleLogger {
       return [[[PTConsoleLogger alloc] init] autorelease];
   }

   - (void)logEntry:(PTLogEntry *)entry {
       //Log the message to the console using NSLog
       NSLog(@"[%@] %@", entry.tag, entry.message);
   }

   @end
   ```

2. Add an instance of the `PTLogger` to the `PTLoggerFactory` to register the instance to receive logging entries:

   ```
   PTConsoleLogger *logger = [PTConsoleLogger consoleLogger];
   // Either use the addLogger method:
   [[PTLogFactory sharedInstance] addLogger:(logger)]

   //Or replace the preceding line with this macro for ease of use
   //PTLogAddLogger(logger);
   ```

Example of filtering logs by using the `PTLogEntry` type:

```
@implementation PTConsoleLogger

+ (PTConsoleLogger *) consoleLogger {
    return [[[PTConsoleLogger alloc] init] autorelease];
}

- (id) init {
    self = [super init];

    if (self) {
        self.logLevel = PTLogEntryTypeInfo;
    }

    return self;
}

- (void)logEntry:(PTLogEntry *) entry {
    //Filtering the entry by log level
    if (entry.type < _logLevel) {
```

```
        return;
    }

    //Log the message to the console using NSLog NSLog(@"[%@] %@", entry.tag, entry.message);
}

@end
```

Adding New Log Messages

To register for listening to logs:

Create a new `PTLogEntry` and add it to `thePTLogFactory`:

You can either manually instantiate a `PTLogEntry` and add it to the `PTLogFactory` shared instance or use one of the macros to accomplish the same task.

Example of logging using the PTLogDebug macro:

```
// The following line creates an instance of PTLogEntry with type PTLogEntryDebug,
// tag "DEBUG_PLAYBACK", and message "Playback has started".
// Then the PTLogEntry is automatically added to the PTLogFactory

PTLogDebug(@"[DEBUG_PLAYBACK] Playback has started");
```

Understanding Failover

.

The PSDK player, rather than the Apple AV Foundation player, provides failover handling.

Failover handling occurs whenever a variant playlist has multiple renditions for the same bit rate and one of the renditions stops working. In this case, the player switches between renditions as described here.

The following example shows a variant playlist with failover URLs of the same bit rate:

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH =700000
http://sj2slu225.corp.adobe.com:8090/_default_/_default_/livestream.m3u8

#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH =700000
http://sj2slu225.corp.adobe.com:8091/_default_/_default_/livestream.m3u8
```

When the PSDK loads the variant playlist, it creates a queue that holds the URLs for all the renditions of the content for the same bit rate. Whenever a request for a URL fails, the PSDK then uses the next URL of the same bit rate from the failover queue. At anyspecific failure time, the PSDK cycles once through all available URLs until it finds one that works correctly or until it has attempted all the available URLs; in the latter case, the PSDK no longer continues attempting to play the content.

Failover occurs only at the M3U8 level. This means:

- For VOD, failover can occur only when it begins attempting to play and not after it has started playing.
- For live streaming, failover can happen in the middle of the stream.

Primetime Player classes summary

You can use the Primetime Player Objective C API to customize the behavior of the player.

Media player classes

These classes describe your media player and its resources.

Class	Description
PTABRControlParameters	Encapsulates all adaptive bit-rate control parameters. Supported parameters are: • minBitRate • maxBitRate • initialBitRate
PTMediaPlayer	Defines the root component for the Primetime Player framework. Applications create an instance of this class to play back a media. This component dispatches notifications to let your application know the status of the player at any given time.
PTMediaPlayerItem	Represents a specific audio-video media.
PTMediaPlayerView	Manages the view component of the Primetime Player framework.
PTMediaProfile	Represents the profile of a single stream in the variant playlist.
PTMediaSelectionOption	Represents an audiovisual media resource to accommodate different language preferences, accessibility requirements, or custom application configurations. Valid option types: • Subtitles (PTMediaSelectionOptionTypeSubtitle) • Alternate audio (PTMediaSelectionOptionTypeAudio) • Undefined (PTMediaSelectionOptionTypeUndefined)
PTSDK	Describes the version of the Primetime SDK and its capabilities.
PTSDKConfig	Exposes PSDK global settings and allows an application to subscribe to custom HLS tags.
PTSiteCatalystTrackingMetadata	Contains property metadata specific to Video Analytics tracking within the PSDK.
PTTextStyleRule	Defines constants that represent text style attribute keys that form the dictionary of rules.

Logging classes

These classes enable you to customize logging.

Name	Description
PTLogEntry	Class. Defines an entry log and holds information about a log message.
PTLogFactory	Class that enables custom logging.
PTLogger	Protocol. The methods required to implement a custom logger for the PSDK.

Metadata classes

These classes provide metadata for advertising, namespaces, and tracking.

Name	Description
AdMetadata PTAdMetadata	Class that provides properties that should be configured for resolving ads for a given media item. All the required properties must be set to configure the player for successfully resolving ads.
PTAuditudeMetadata	Class that extends `PTAdMetadata` specifically for Ad Decisioning. Provides properties to be configured for resolving Ad Decisioning ads for a given media item. You must set all the required properties, including zone ID, media ID, and ad server URL, to configure the player for successfully resolving ads.
PTMetadata	Defines the base class for configuring all available metadata for your player. Individual components extend this class to provide additional accessors for key-value metadata associated with PSDK objects.
PTTimedMetadata	Class that represents a custom HLS tag in the stream.
PTTrackingMetadata	Defines a base class for all tracking and analytics related metadata.

Notification classes

These classes describe messages about errors, warnings, and some activities that the PSDK issues for logging and debugging purposes.

Class name	Description
PTMediaError	Class that describes the notification for an error that causes the player to stop playback. This is a *PTNotification* of notification type ERROR.
PTMediaPlayerNotifications	Lists all the notifications dispatched by the Primetime Player framework.
PTNotification	Class that provides informational messages, warnings, and errors. Encapsulates the object representation of a single notification event within *PTNotificationHistory*.
PTNotificationHistory	Class that stores a log of notification objects. A circular list of *PTNotificationHistoryItem* objects that provides access to a notification events history list. That is, it maintains a list of elements, each element containing a separate instance of the *PTNotification* class.
PTNotificationHistoryItem	Class that defines an entry in the circular list in *PTNotificationHistory* and holds the notification and its timestamp.

QoS classes

These classes provide information that help you to determine how well the player is performing.

Name	Description
PTDeviceInformation	Provides information about the platform and operating system on which the PSDK runs: • Version of the platform OS • Version number of the PSDK library • Device's model name • Device manufacturer's name • Device UUID • Width/height of the device screen
PTPlaybackInformation	Provides information on how the playback is performing. This includes the frame rate, the profile bit rate, the total time spent in buffering, the number of buffering attempts, the time it took to get the first byte from the first video fragment, the time it took to render the first frame, the currently buffered length, and the buffer time.
PTQoSProvider	Provides essential QoS metrics for both playback and the device.

Timeline classes

These classes provide information about the timeline of the particular media, including the placement of ads.

Name	Description
PTTimeline	Class that represents the timeline of the content, including ad breaks.

Timeline advertising classes

These classes provide information about ads that occur within a timeline.

Name	Description
PTAd	Class that defines the Ad abstraction and holds all ad information. It is defined by a unique ID, a duration, and a MediaResource. The MediaResource contains the URL where the actual ad content resides. Represents a primary linear asset spliced into the content. It can optionally contain an array of companion assets that must be displayed along with the linear asset.
PTAdAsset	Class that represents an asset to be displayed. Represents an asset to be displayed.
PTAdAvailInfo	Describes an ad opportunity within the stream.
PTAdBannerView	Displays a banner asset. Your application must create a new instance of this utility class, set the banner asset, and add it to a view. The impression and click tracking for the banner is internally managed by this class.
PTAdBreak	Class that gives a unified view on several ads that will be played at some point during playback. Represents a continuous sequence of ads spliced into the content.

Name	Description
PTAdClick	Class that represents a click instance associated with an asset. This instance contains information about the click-through URL and the title that can be used to provide additional information to the user. Represents a click instance associated with an asset. This instance contains information about the click-through URL and the title that can be used to provide additional information to the user.
PTAdHLSAsset	Represents an HLS asset for a break.
PTAdHLSAssetLoaderDelegate	Protocol for a class to be a listener to `PTAdHLSAsset` loading callbacks.

Digital rights management classes

These classes provide information about DRM activity.

Class	Description
`PTDRMMetadataInfo`	Represents a specific DRM metadata instance.

Notification codes

The PSDK notification system includes various classes of notification events produced by the PSDK.

These notification events expose numerical codes that are grouped into ranges of integer values. The ranges provide two levels of classification.

Notification events are grouped in the following top-level classifications:

• Error notification events, from 100000 to 199999

• Warning notification events, from 200000 to 299999

• Information notification events, from 300000 to 399999

Inside each top-level range, subranges further classify notifications. Each subrange contains up to 1000 values, so each top-level range can contain up to 100 second-level subranges.

ERROR notification codes

Code	Name	InnerNotification	Metadata Keys	Comments
DRM				
100000	DRM_ERROR		MAJOR_DRM_CODE MINOR_DRM_CODE DESCRIPTION	
Playback				
101000	PLAYBACK_ERROR			
101001	NATIVE_PLAYBACK_ERROR	None	DESCRIPTION INTERNAL_ERROR URL	

Code	Name	InnerNotification	Metadata Keys	Comments
101008	SEEK_ERROR	None	DESCRIPTION	An error has occurred while performing a seek operation.
101009	PAUSE_ERROR	None	None	An error has occurred while performing a pause operation.
101101	AUDIO_TRACK_CHANGE_FAIL	PLAYER_NOT_READY	None	
Invalid resource				
102000	INVALID_MEDIA_PLAYER_ITEM	None	None	
Ad processing				
104001	AD_RESOLVER_METADATA_INVALID	AD_NOT_INSERTED	None	Ad resolving failed due to invalid ad-metadata format.
104005	AD_INSERTION_FAIL	AD_NOT_INSERTED	None	Ad resolving phase has failed.
104006	AD_UNREACHABLE	None	None	
Native				
106000	NATIVE_ERROR	None	INTERNAL_ERROR	A low-level iOS error occurred.
Configuration				
107002	SET_CC_VISIBILITY_ERROR	None	None	An error has occurred while attempting to change the visibility of the CC tracks.
107003	SET_CC_STYLING_ERROR	NATIVE_ERROR	None	An error has occurred while attempting to change the styling options for the CC tracks.
iOS specific				
170000	AD_HLS_VERSION_INCOMPATIBLE	AD_INSERTION_FAIL	None	The HLS version of the ads are highter than the HLS version of the content.
170001	ARGUMENT_ERROR	None	None	Argument error
170002	M3U8_PARSER_ERROR	None	DESCRIPTION	Could not parse m3u8.

Code	Name	InnerNotification	Metadata Keys	Comments
170003	WEBVTT_PARSER_ERROR	None	None	Could not parse Webvtt.
170004	HLS_SEGMENT_ERROR	None	DESCRIPTION URL INTERNAL_ERROR	Segment exceeds specified bandwidth for variant.
170005	MBR_MEDIASEQUENCE_OFFSYNC	None	None	The media sequence number is not on sync on all the HLS streams of this MBR.
170006	MISSING_FILE_ERROR	None	DESCRIPTION URL INTERNAL_ERROR	Missing file or not responding. HTTP 404: File not found.
170007	AD_EMPTY_RESPONSE	AD_INSERTION_FAIL	None	Could not retrieve ads. Empty response.
170008	AD_TIMEOUT	AD_INSERTION_FAIL	None	Could not retrieve ads. Timeout error.
170009	SUBTITLES_TRACK_CHANGE_FAIL	PLAYER_NOT_READY	None	Error while changing the subtitles track.
170010	SITECATALYST_ERROR	None	DESCRIPTION	Site catalyst error. See description.
170011	AD_TARGETDURATION_INCOMPATIBLE	AD_INSERTION_FAIL	None	The TARGET DURATION of the ad is higher than the TARGET DURATION of the content.

WARNING notification codes

Most warnings contain relevant metadata (for example, the URL of the resource that failed to be downloaded). Some warnings contain metadata to specify whether the problem occurred in the main video content, in the alternate audio content, or in an ad.

Code	Name	InnerNotification	Metadata Keys	Comments
iOS specific				
270000	PLAYER_NOT_READY	None	DESCRIPTION	
270001	AD_NOT_INSERTED	None	None	AD was not inserted on the stream.

Code	Name	InnerNotification	Metadata Keys	Comments
270002	AD_HLS_AUDIO_ONLY_MISSING	AD_NOT_INSERTED	None	Ad does not contain Audio Only Stream
270003	AD_HLS_MATCHING_BITRATE_MISSING	AD_NOT_INSERTED	None	No matching ad stream found for content's current bitrate.
270005	AVASSET_FAILED_TO_CREATE	PLAYBACK_ERROR	None	Error at creating the AVAsset.
270006	SITECATALYST_WARNING	None	DESCRIPTION	Warning: See sitecatalyst warning description.
270007	NETWORK_ERROR	None	URL	Error getting data from the network.

INFO notification codes

Code	Name	InnerNotification	Metadata Keys	Comments
Playback				
300000	PLAYBACK_START	None	None	Notifies that playback has started
300001	PLAYBACK_COMPLETE	None	None	Notifies that playback has completed
300002	SEEK_START	None	None	Notifies that a seek operation was initiated
300003	SEEK_COMPLETE	None	None	Notifies that a seek operation has completed.
300005	PLAYER_STATE_CHANGE	None	None	Notifies that the player state has changed. When state is ERROR, the inner notification is the error notification object that triggered the switch to the ERROR state.
Adaptive bit rates (ABR)				
302000	BITRATE_CHANGE	None	BITRATE	Notifies that the bitrate of the video has changed
Late-binding audio (LBA)				
304000	AUDIO_TRACK_CHANGE	None	None	Notifies that the audio track has been changed.
Subtitles				

Code	Name	InnerNotification	Metadata Keys	Comments
`307000`	`SUBTITLES_TRACK_CHANGE`	None	None	Notifies that the subtitles track has changed

Glossary

A

ABR

Adaptive bit rate.

Based on the bandwidth conditions and the quality of playback (frame rate), the video engine automatically switches the quality level (bit rate) to provide the best playback experience.

ABR algorithm

The algorithm that determines which bit rate the client bandwidth can handle.

adaptive set

A set of renditions, typically from the same CDN. Within a set, renditions represent different bit rates.

AVE

Adobe Video Engine.

C

cue point

An #EXT tag in an M3U8 file that the PSDK uses for splicing ads into a live stream.

D

DRM

Digital rights management.

F

fragment

One segment of one representation of the media presentation.

H

HDS

HTTP Dynamic Streaming

This technology is from Adobe. The format works in OSMF 1.6 and later. HTTP Dynamic Streaming reproduces much of the functionality of RTMP delivery, providing the publisher a choice in delivery options. The primary benefit that HTTP offers is its ability to cache content, which is important for enterprise customers who deploy internal caching systems to optimize network usage.

HLS

HTTP Live Streaming

This Apple technology is a required format for delivery to Apple devices. It works in HTML5 browsers, AIR for iOS player, and native iOS apps.

I

iframe

An HTML iframe. An inline frame places another HTML document in a frame. An inline frame can be the target frame for links (URLs) defined by other elements, and it can be selected by the user agent as the focus for printing, viewing its source, and so on. The content of the element is used as alternative text to be displayed if the browser does not support iframes.

L

linear video

Linear video is streamed video with advertisements already stitched into the stream. An example of linear video could be a programmed television show.

live video

Live video is video that is being streamed from a live event. Advertisements are typically overlaid on live video. An example of live video could be a live sporting event.

M

main content

Represents the movie or streaming video.

MBR

Multipe bit rate.

N

national ads

Ads that are placed in the main content, but not targeted.

P

PHDS

Protected HTTP Dynamic Streaming.

PHLS

Protected HTTP Live Streaming.

Q

QoS

Quality of service.

R

rendition

One representation of the media presentation.

RTA

Real-time analytics.

S

SBR

Single bit rate.

T

targeted ads

Ads that are served to a user based on received information about that user.

V

VOD

Video on demand.

Video that you watch at your convenience. An example of VOD is a video that you can download on your device from a video publishing service.

Copyright