Android SDK v5

An Akamai Professional Services Solution

User Guide

Updated: 29-May-15
Version: 5.24.0

TABLE OF CONTENTS

OVERVIEW

About the Android SDK

Akamai Android SDK is a native binary module for Android OS that enables developers to write Android applications that are able to play HLS streams.

Features

- Software decoding and Hardware accelerated playback
- Native basic mode, where the native Android player is used
- Plays both Video-On-Demand [VOD] and LIVE streams
- Renders multi bit-rate and single bit-rate content
- Easy-to-integrate. User control that can be embedded into any Android activity layout
- Complete API to control the playback and get analytic stats
- Support for DVR and seeking
- Support for audio-only playback in background (Radio applications)
- Built-in support for secure streaming
    - AES content encryption
    - Token authentication support
- Integrated with Sola Analytics and HDClient Akamai products.

ANDROID SDK REQUIREMENTS

Pre-requisites

- Android SDK requires Android OS 1.6 or above (4.0 minimum required for Hardware Advanced)
- HTTP Live streaming streams
    - Supported video codecs
        - H.264
    - Supported audio codecs
        - AAC
    - Audio/Video encoding must follow Apple's recommendations as specified in http://developer.apple.com/library/ios/#technotes/tn2224/_index.html in conjunction with Android's, http://developer.android.com/guide/appendix/media-formats.html#recommendations for maximum compatibility.

Package contents

The Android SDK is distributed as a ZIP file package. The package contains the following folders:

Folder	Description
doc	Full developer documentation
samples	Samples source code
libs	Set of libraries composing the SDK

INTEGRATING THE ANDROID SDK

Adding it to your application layout

Extract the contents of the SDK package zip file. Add the Android SDK libraries to your project. It should be enough to copy the SDK Libs folder content into the libs folder of your project.

Instantiate the VideoPlayerContainer object (equivalent of the Android VideoView object) in your activity layout:

```
<com.akamai.media.VideoPlayerContainer android:layout_width="fill_parent"
    android:id="@+id/playerViewCtrl" android:layout_height="fill_parent"
    android:orientation="vertical" android:gravity="center"
    android:layout_gravity="center" />
```

Use the workingMode property of the VideoPlayerContainer object for setting the video decoding method to use: *software*, *hardware*, *hardware advanced* or *native basic*. Please, keep reading for more information on the different decoding modes.

In the onCreate method of your activity, sets the decoding mode to be used by the Android SDK and then get a reference to the VideoView object.

```
@Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        // Get a reference to the VideoContainer object
        mVideoContainer = (VideoPlayerContainer) findViewById(R.id.playerViewCtrl);

        // Set the decoding mode used by the Android SDK video contol
        mVideoContainer.setMode(MODE_SOFTWARE);

        // Get a reference to the VideoView object. NOTE: use .getAudioPlayer() for
        // audio-only playback
        mVideoObject = mVideoContainer.getVideoPlayer();
    }
```

The decoding modes available in the Android SDK are:

- **Hardware Advanced mode**. Recommended mode; provides fast startup times, hardware acceleration and support for automatic bitrate switching (adaptive bitrate). Compatible with Android 4.0 devices and above.
- **Software decoding**. Android SDK will use software based codecs for decoding the video/audio of the stream. This means CPU consumption will be higher than when using hardware based decoding. As an advantage, this method is free of any restriction of the device regarding video encoding limitations. Software based decoding is the method selected by default. Automatic bitrate switching is supported. Compatible with Android 1.6 devices and above.
- **Hardware decoding**. Android SDK will use the H.264 decoder chipset present in any Android device for decoding video/audio of the stream. Quality offered by this method is much better than the one you can get using software based decoding although this method has the same limitations of the H.264 decoding chipset: not supporting other H.264 profile than Baseline. Automatic bitrate switching is not supported in this mode. Compatible with Android 1.6 devices and above.

- **Native basic mode**. Using this mode, the Android SDK will directly use the native player implemented by Android. Due to the different implementations of the native player in the different versions of Android, this mode will not be available on multiple devices. Please, refer to the Native basic mode restrictions section of this document for more information.

- **Auto mode**. This mode follows the next logic to determine the decoding method to be used:

    - **If Android device is 4.0 or above, Hardware Advanced mode is selected.**
    - **If Android device is 3.X or below:**
        - **If the number of cores inside the CPU of the device is 2 or higher, Software mode is selected**
        - **If the CPU of the device contains a single core, Hardware mode is selected.**

Enabling Hardware decoding mode

To enable Hardware decoding, the *com.akamai.media.hls.AkamaiHLSService* service must be declared in the Android Manifest file inside the application section:

```
<service android:name="com.akamai.media.hls.AkamaiHLSService"></service>
```

Playing a stream

Use the **playUrl()** method of the VideoPlayerView object to start playing a stream:

```
// Play the stream
    mVideoView.playUrl("http://devimages.apple.com/iphone/samples/bipbop/bipbopall.m3u8");
```

Note: use **playAudioUrl()** for audio-only streams

Note: more information in the "Integration Video Tutorial":
http://projects.mediadev.edgesuite.net/customers/akamai/android/HelloHLS-MakingOf-Edited.mp4

BASIC PLAYBACK METHODS

Below are described the methods you can use in your application for supporting basic playback commands. More information is available in the online API documentation:
http://projects.mediadev.edgesuite.net/customers/akamai/android/doc/index.html

Methods

playUrl(stream_url)
Plays a stream indicated in the parameter stream_url.

playUrl(stream_url, position)
Plays a stream indicated in the parameter stream_url, starting at the specified position (in seconds).

playAudioUrl(stream_url)
Plays an audio-only stream indicated in the parameter stream_url. This method should be used when developing an application that does not implement a video layer; for example, a service to reproduce audio streams in background.

playAudioUrl(stream_url, position)

Plays an audio-only stream indicated in the parameter stream_url, starting at the specified position (in seconds).

pause()
Pauses the playback. Doesn't work when player is doing a seeking or bitrate switching operation.

resume()
Resumes the playback. Doesn't have any effect if playback is not in pause state.

seek(position)
Does a seek operation to the specified time position. Time position can be indicated as an absolute position (0 is start of the stream) or as a relative position (0 is current playback position).
This method shouldn't be called until the PLAYER_EVENT_TYPE_START_PLAYING event is raised.

setFullScreen(mode)
Enables/Disables fullscreen mode. When fullscreen mode is enabled Android SDK will render the video content using as much space as possible of the VideoPlayerView object. This doesn't mean the SDK is going to resize the VideoPlayerView object to fill the entire screen. If this is required, application should be responsible of doing this.

getDuration()
[int] Returns the total duration of the media resource in seconds. For live streams return 0 seconds.

getTimePosition()
[int] Returns the current playback position, in seconds. For live streams returns seconds since the playback started and its value reset whenever the user does a seeking or bitrate change operation.

getTimePositionMS()
[long] Returns the current playback position, in milliseconds. For live streams returns milliseconds since the playback started and its value reset whenever the user does a seeking or bitrate change operation.

isPlaying()
[boolean] Returns true if playback is in progress; false otherwise.

isPaused()
[boolean] Returns true if playback is in pause state; false otherwise.

isSeeking()
[boolean] Returns true if a seeking operation is in progress; false otherwise.

isFullScreen()
[boolean] Returns trus if VideoPlayerView object is in fullscreen mode; false otherwise.

isFinished()
[boolean] Returns true if the playback has finished (reached the end of the stream); false otherwise.

isLive()
[boolean] Returns true if playing a live stream; false otherwise.

isError()
[boolean] Returns true if VideoPlayerView object is in error state; false otherwise

MANAGING PLAYBACK EVENTS

The VideoPlayerView object fires events that can be captured by your application to know the status of the playback and adapt your user interface.

To subscribe your activity to the VideoPlayerView object you should use the method setEventListener and implement modify the activity to implement the interface IPlayerEventsListener.

Important: VideoPlayerView object could fire events from a thread different than the UI thread. This means you SHOULD NOT assume you can modify the user interface directly in your events listener method. A UI handler should be used for this.

Methods

setEventsListener(listener)
Subscribes the listener object to the events fired by the VideoPlayerView object

Listener should implement the methods:
onPlayerEvent(eventType)
Called when event of type eventType occurs.

onPlayerExtendedEvent(eventType, arg1, arg2)
Called when extended events of type eventType occurs. Meaning of the parameters arg1 and arg2 depends on the event fired.

Events

PLAYER_EVENT_TYPE_LOADING
Dispatched to indicate the stream is being loaded

PLAYER_EVENT_TYPE_START_PLAYING
Dispatched as soon as playback begins. Enables seek operations.

PLAYER_EVENT_TYPE_POSITION_UPDATE
Dispatched to let the application know the time position has changed.

PLAYER_EVENT_TYPE_FINISHED
Dispatched to let the application know the playback has finished.

PLAYER_EVENT_TYPE_ERROR
An error occurred while trying to play the stream.

PLAYER_EVENT_TYPE_START_REBUFFERING
VideoPlayerView doesn't have enough data for continuing the playback. Playback will be paused until enough data is in the buffer for continuing with the playback smoothly.

PLAYER_EVENT_TYPE_END_REBUFFERING
Indicate the end of a rebuffering event.

PLAYER_EVENT_TYPE_SWITCH_REQUESTED
Dispatched to let the application know a bitrate switch was requested.

PLAYER_EVENT_TYPE_SWITCH
VideoPlayerView has changed the bitrate used for the playback.

PLAYER_EXTENDED_EVENT_BANDWIDTH_MEASURE
Extended event. Give the application information about the client bandwidth. Arg1 parameter will contain the measure client bandwidth in bps. Arg2 parameter will contain the bitrate recommended for the playback. Arg2 is calculated by the VideoPlayerView object knowing the current bandwidth.

PLAYER_EVENT_SEEKING_SUCCEDEED
Extended event. Dispatched when a seeking event finished successfully. Arg1 contains the resulting seeking position.

BITRATE SWITCHING

Android SDK software and hardware advanced modes support automatic bitrate switching. Hardware Advanced mode is compatible with Android devices 4.0 and above.

Android SDK hardware version, compatible with Android device 1.6 and above, doesn't support automatic bitrate switching but it offers a full set of methods to let the application configure the bitrates used during the playback:

Methods

setHLSStartingAlgorithm(mode)
Sets the algorithm used for selecting the initial bitrate of the playback. There are two options available:

- HLS_STARTING_ALGORITHM_APPLE
  Algorithm will be similar to the one used by iDevices: Choose as the initial bitrate the first bitrate defined in the playlist.
- HLS_STARTING_ALGORITHM_AKAMAI
  Algorithm designed by Akamai optimized to cover the maximum number of devices. Chooses the highest bitrate below 300 Kbps. The limit of 300 Kbps can be modified using the method setAkamaiAlgorithmValue(int bitrate) – in bps .

setMaxBitrate(bitrate)
Sets the maximum bitrate available for the playback.

When using HLS_STARTING_ALGORITHM_AKAMAI algorithm for selecting the initial bitrate, the SDK will choose as the initial bitrate the highest one below the one defined with this method.

switchBitrateUp()
Switches to the higher bitrate.

swithBitrateDown()
Switches to the lower bitrate.

setBitrateToPlay(value)
Software Mode: Switches to the highest bitrate below or equal to the one passed in the parameter bitrate.
Hardware & Hardware Advanced mode: sets the bitrate index to play

setStartingBitrateIndex(index)
Sets index of the bitrate to be used when playback starts. Index 0 will be assigned to the first bitrate defined in the master playlist (typically master.m3u8), Index 1 to the second bitrate, Index 2 to the third bitrate, etc.

getBitratesCount()
[int] Returns the number of bitrates available for the current stream.

getCurrentBitrate()
[long] Returns the bitrate used for the playback in bps.

getBitrateByIndex(index)
[long] Given a bitrate index, return bitrate in bps.

DVR MANAGEMENT

Android SDK is able to navigate (do seeking operations) through the stream playlist (.m3u8 with the list of segments) even for live streams.

It is important to know that the length of the DVR in HLS is defined by the number of segments included in the playlists. For example, if we are using 10 seconds fragment length and it is a requirement to have a DVR of 3 hours, the playlists should reference a total of 1080 segments ((3 hours * 60 minutes/hour * 60 seconds/minute) / 10 seconds/fragment). This makes the playlist to be very big when using long DVR. For helping on this, Android SDK hardware version support gzip encoding for requesting the playlists via HTTP.

Methods

getDVRLength()
[long] Returns the length of the DVR in seconds

getPositionInDVR()
[int]Return the current playback position relative to the DVR. Its value will be 0 when playing the oldest content in the DVR, and will be getDVRLength() when playing in the live position. For VOD streams, returns 0.

getTimePositionAsDate()
[Date] Returns the current playback position as an absolute time (localtime zone is used).

seekToLive()
Seeks to the live position. Only working for live streams.

RETRIEVING PLAYBACK INFORMATION

Android SDK has the following methods for retrieving information about how is going the current playback:

Methods

getBufferingPercentage()
[int] Whenever a rebuffering event is happening this method returns the percentage of the buffer that is currently filled. Its value goes from 0 to 100.

getBytesLoaded()
[int] Returns the total number of bytes downloaded by the Android SDK to play the current stream. Its value is reset to 0 each time a call to playUrl is done.

getLastHttpErrorCode()
[int] Returns last error code (200, 403, 404, etc) returned by the HTTP engine responsible of downloading the content of the stream (both playlists and segments). For example, when playing a geoblocked stream this methods will return the value 403 to let the application know the user doesn't have rights to watch the content.

getRebuffers()
[int]Number of rebuffer events that occurs along the playback. Its value is reset to 0 each time a call to playUrl is done.

getRebufferingTime()
[double] Number of seconds the player has been paused filling the playback buffer.

getVideoWidth()
[int] Returns the width, in pixels, of the loaded stream.

getVideoHeight()
[int] Returns the height, in pixels, of the loaded stream.

getVersionDescription()
[String] Returns extended information about the Android SDK version used.

getStreamsInfo()
[String] Returns information about the stream in text format, including the list of bitrates.

about()
[String] Returns the about text. You have to include this text in the About dialog of your application.

AUDIO-ONLY PLAYBACK

Playback of audio-only streams is quite similar to video playback. There are two implementation changes:

- Use **VideoPlayerContainer.getAudioPlayer()** instead of getVideoPlayer()
- Use **VideoPlayerView.playAudioUrl()** instead of playUrl();

The recommended decoding mode for audio-only playback is AUTO. This will use Hardware Advanced mode for Android 4.1.2 devices and above, and Hardware for lower versions.

Please, see the **AudioPlayer** example project inside the samples/ folder for details.

SOLA ANALYTICS INTEGRATION

Akamai Sola Analytics (previously known as Media Analytics) provides detailed client-side and server-side reporting services. Server-side reports typically contain audience engagement and content usage information. The client-side reports provide information on user interaction and bandwidth. The Android SDK supports Client-side Media Analytics (CSMA).

Overview

CSMA support is enabled through the CSMA component – a plug-in used by the player to gather various statistics that would be used to generate the resorts. The CSMA plug-in uses a configuration XML file to identify the statistics to be collected and log them in a specific location. The configuration XML file path is provided during provisioning.

Using Sola Analytics

setMediaAnalyticsConfigUrl(configUrl)
Sets the Sola Analytics url config (XML file). This configuration file path is provided during provisioning of the Sola Analytics report pack.

setMediaAnalyticsCustomData(key, data)
Reports a custom dimension (dimension name = key) from the player.

setMediaAnalyticsViewerId(viewerId)
Sets the viewer ID for the Viewer Diagnostics module in Sola Analytics.

LOGGING AND DEBUGGING

Methods

setLogEnabled(value)
Enables/disables log traces (Android DDMS log traces). Log is disabled by default.

setDebuggingActive(value)
Activates/deactivates the Android SDK mechanism for getting debug information from the player. When enabled, the Android SDK will send debug information to a remote server after each playback. The remote server is defined using the method setDebugUrl.

setDebugUrl(url)
When debugging mode is active each time the player finishes playing a stream it sends a debug report to the url defined using this method. The url parameter allow three variables that will be replaced dynamically just before sending the information:

- %DEVICEID. Device ID. It could be NULL if the device doesn't have a specific device ID.
- %TIME. Time when the playback finished, in epoch UNIX format.
- %ERRORCODE. Last error code (0 if playback was successful, otherwise a value < 0).

Example of valid url:
http://debug.android.testing.com/postDebug.php?id=%DEVICEID&time=%TIME&error=%ERRORCODE

ANDROID SDK LICENSING

Android SDK v4 includes a protection mechanism based on license codes. Licenses are generated by Akamai and are only valid for one specific application. In case you run your application without setting the license, Android SDK will not work.

Methods

setLicense(license)
Sets the Android SDK license of your application. License will be provided by Akamai.

isLicenseExpired()
[Boolean] Returns true if the set license has expired or if it is not valid. If the license has expired Android SDK works in trial mode.

getLicenseExpirationDate()
[Date] Returns the license expiration date. After the expiration date the SDK won't be able to play any video.

getLicensePackageName()
[String] Returns the package name for which the license was generated. If the package name is different than the package name of the application, the SDK won't be able to play any video.

NATIVE BASIC MODE RESTRICTIONS

As it was mentioned before, Android SDK v4 introduces a new native basic mode which by-pass the stream playback to the native media player of the Android system. Due to this, many of the methods and events implemented for Software, Hardware and Hardware Advanced decoding modes are not available, as the Android system does not provide this information to the SDK.

This is a summary of the different restrictions per method/event existing in the API:

isLive()

Not available, "false" is always returned.

onPlayerExtendedEvent(eventType, arg1, arg2)

Extended events are not available

PLAYER_EVENT_TYPE_START_REBUFFERING

Availability of this event depends on the device and Android version.

PLAYER_EVENT_TYPE_END_REBUFFERING

Availability of this event depends on the device and Android version.

PLAYER_EVENT_TYPE_SWITCH_REQUESTED

Not available.

PLAYER_EVENT_TYPE_SWITCH

Not available.

PLAYER_EXTENDED_EVENT_BANDWIDTH_MEASURE

Not available.

PLAYER_EVENT_SEEKING_SUCCEDEED

Android system documentation states this event is implemented, but it does not work on most of devices.

setHLSStartingAlgorithm(mode)

Not available.

setMaxBitrate(bitrate)

Not available.

switchBitrateUp()

Not available.

swithBitrateDown()

Not available.

setBitrateToPlay(bitrate)
Not available.

setStartingBitrateIndex(index)
Not available.

getBitratesCount()
Not available.

getCurrentBitrate()
Not available.

getBitrateByIndex(index)
Not available.

getDVRLength()
Not available.

getPositionInDVR()
Not available.

getTimePositionAsDate()
Not available.

seekToLive()
Not available.

getBufferingPercentage()
Not available.

getBytesLoaded()
Not available.

getLastHttpErrorCode
Not available.

getRebuffers()
Not available.

getRebufferingTime()
Not available.

getStreamsInfo()
Not available.

setMediaAnalyticsConfigUrl(configUrl)

Not available.

setMediaAnalyticsCustomData(key, data)

Not available.

TROUBLESHOOTING

Video frozen while audio is playing

This could happen when starting the playback or just after a bitrate change. This occurs when the profile used for encoding the video is not supported by the hardware of the Android device. As we are using the hardware of the device we are limited to the profiles/formats supported by it.

Note: This can also happens when user select to play the only audio bitrate. In this case we are getting the right behavior, no video is rendered (video frozen) and audio is played.

Solution:

The most of the Android devices only support H.264 Baseline Profile so, for maximum compatibility, be sure you use this profile when encoding your streams. As a general rule, please follow Apple's recommendations as specified in http://developer.apple.com/library/ios/#technotes/tn2224/_index.html in conjunction with Android's, http://developer.android.com/guide/appendix/media-formats.html#recommendations for maximum compatibility.

Hardware decoding mode does not work

This might occur when the AkamaiHLSService is not declared in the application section of the Android Manifest XML file.

```
<service android:name="com.akamai.media.hls.AkamaiHLSService"></service>
```

Audio/Video sync issues

This happens when audio and video timestamps are not correctly aligned. Although the Android SDK tries to fix big misalignments issues generated by the encoder, it is possible that many consecutive small misalignments generate audio/video sync issues during the playback.

Solution:

Please, check your encoder configuration to try to fix this issue. If the issue is still there, contact with Akamai and we will analyze the stream to find out where is the problem.

Playback stopped after some minutes

The reason behind this is you are not setting a valid license using the VideoPlayerView method setLicense.

Solution:

Ask Akamai for getting an Android SDK license.

Stream does not work after the Activity has been paused and resumed

The SDK does not automatically manage the life cycle of the activity.

Solution:

Playback should be stopped in the onPause() event of the activity, and resumed at the same position in the onResume() event. Example:

```
@Override
protected void onPause()
{
    if (mVideoView.isPlaying() || mVideoView.isPaused())
    {
        mCurrentPosition = mVideoView.getTimePosition();
        mWasPlaying = true;
        mVideoView.stop();
    }

    super.onPause();
}

@Override
protected void onResume()
{
    if (mWasPlaying)
    {
        mVideoView.playUrl(mUrl, mCurrentPosition);
        mWasPlaying = false;
    }

    super.onResume();
}
```

Video displays pixelation or ghosting effect after seeking in Hardware Advanced

Video keyframes inside the TS segments of the HLS stream are not aligned with the duration of the segments. There are two different ways to fix this issue:

1. Fix the keyframe alignment at the encoder level – for example, setting a fixed keyframe interval of 2 seconds for 10 seconds segments.
2. Configure the SDK to force a format change after every seek operation:

```
mVideoView.setForceFormatChange(true);
```

The Akamai Difference

Akamai® provides market-leading, cloud-based services for optimizing Web and mobile content and applications, online HD video, and secure e-commerce. Combining highly-distributed, energy-efficient computing with intelligent software, Akamai's global platform is transforming the cloud into a more viable place to inform, entertain, advertise, transact and collaborate. To learn how the world's leading enterprises are optimizing their business in the cloud, please visit www.akamai.com and follow @Akamai on Twitter.

Akamai Technologies, Inc.

U.S. Headquarters
8 Cambridge Center
Cambridge, MA 02142
Tel 617.444.3000
Fax 617.444.3001
U.S. toll-free 877.4AKAMAI
(877.425.2624

www.akamai.com

International Offices
Unterfoehring, Germany
Paris, France
Milan, Italy
London, England
Madrid, Spain
Stockholm, Sweden
Bangalore, India
Sydney, Australia
Beijing, China
Tokyo, Japan
Seoul, Korea
Singapore