SMARTCLIP SDK

Version history

Document version	Corresponding SDK-version	Corresponding JS core	editor
1.0	Native: 1.2.4 (iOS)/1.2.2 (Android)	4.6.0	Karl Szwillus

Contents

Introduction

The smartclip SDK is a development kit for displaying ads in your native Android or iOS App. It supports both out-stream and in-stream use cases. Out-stream in context of a native app means placing ads among various kinds of non-video content. This can be achieved in several ways, let it be fixed layouts or dynamically generated lists. Other cases include advertising of non-video streaming or gaming use-cases. For the In-Stream use case the smartclip player is placed on top of the video player used for showing the actual content. It features a sequencing map that can be configured to show ads at pre-defined positions in your video.

Features

AdView-Settings and configuration

In the SDK the following parameters can be controlled by the native SDK:

- The Ad-Tag
- Setting a header text for the ad player
- Defining a custom skip offset (for users)
- Setting an Advertiser ID as provided by Apple (IDFA) or Google (Advertising Identifier)
- Request bitrate settings
- Control the ad start/pause default behavior
- Overwrite CSS settings of default player
- Overwriting the Bundle-ID (for debugging and testing purposes)

Setting a header text for the ad player

Compliance rules and legal regulations require to show an identifier for video adverts in your native app as well as in web sites. The default term for this behavior is the unlocalized String "Advertisement" in the upper left corner of your video view. This can be changed and adjusted according to your usecase.

Defining a custom skip offset (for users)

If your business model allows skipping the ads for your user after a certain point of time you can set a custom skip offset with this setting

Setting an Advertiser ID as provided by Apple (IDFA) or Google (Advertising Identifier)

In case you are using the IDFAor the Advertising Identifier in your apps or plan to do so, you can also hand it over to the SDK and the ad server. Since the field is not validating it is also possible to generate any other identifier within your app, which is used as a means of tracking. The SDK will not try to fill this information of the Advertising tracker. Doing so requires special attention to either the permissions on Android or during the App submission process on iOS.

Request bitrate settings

It is possible to override the settings for the desired bitrate. Please note that the SDK already will take parameters like screen size, device category (phone/tablet) and wifi vs. mobile connection into account when calculating an optimized bitrate. Control the ad start/pause default behavior. Using the Smartplayer component on your website or mobile App gives you certain parameters like start or offscreen behavior. There is a sensible default setting, but if your specific App usecase has different requirements you are able to adapt to this.

Overwrite CSS settings of default player

The main component used for displaying ads is an HTML5-based Video view. That allows some modifications of the underlying styles of the player component. Elements like scrollbars, titles or other controls can use different icons/colors or typography but adding an individual stylesheet to your AdView component.

Use different Bundle-ID (for debugging and testing purposes)

The distribution of HTML5 and JavaScript components is associated with the bundle identifier. Changing it will result a different behavior. *Only use this setting after talking to your smartclip contact about how to utilize it.* This setting should not be used in production to redirect to any other than your own Bundle-Identifier or App ID.

Extended clickthrough handling

Usually the ad view component handles clicks on the videos if a landing page URL is provided. With the extended click handling setting this behavior can be controlled by the native app.

Callbacks

During an ad slot or ad session you can access a number of state information by registering a listener/delegate to handle callbacks. This will allow your own app to interact properly with the advertising content and integrate your own control flow.

onPrefetchCompleteCallback

Using prefetching allows your application to seamlessly integrate ad-calls with your content by perfecting the data and stopping ad playback until the conditions to actually display the ad are reached. The handling of this needs some extra work on the UI and a thorough use of all callbacks combined.

onStartCallback

This callback signals the start of an Ad slot with the app. This call is expected to happen shortly after initializing and/or a positive visibility check, depending on your Adview's configuration.

onEndCallback

This callback signals the end of an ad slot. In In-Stream scenarios it can be utilized to create a more refined interaction with all kinds of preroll-content.

onCappedCallback

The onCappedCallback signalizes that there is no suitable content for the ad tag at the time of the request. It allows you to use native fallback mechanism from outside your ad server to react to this situation.

onEventCallback

If there is a registered listener for the onEventCallback it will receive all status transitions that the Adview or the ad slot goes through. It's main purpose is to use it for native reporting of different kinds.

onClickthru

The call for onClickthru is a callback that allows your app to change the control flow of the clickthrough to the landing page of the ad. It can be used to ask for confirmation or show further options than Yes/No. Adding additional click trackers is possible on this call, too. In case the extended click handling is activated this callback needs some extra attention (s ee Implementation Guide).

Retrieving ad and display information

With the onEventCallback your application will get information about the current state of the ad player. It starts with the signal of AdSlot Start, the interpretation of the given answer from the ad server and will iterate through the ads that are displayed within one request. Callbacks follow the VAST standard signalling quartiles and additional information on the type of the ads. In addition to following the continuous stream of events it also possible to call getAdInfo on your SCAdview, if you need the information for reporting or any decisions within your application logic. Please refer to the JavaDoc/JazzyDoc parts to get information on all fields/methods available for the AdInfo.

Sequencing and ad variants

There are some advanced options available to display more complex scenarios, especially for In-Stream settings, for instance to play multiple ads in one single ScAdView. This can be achieved through two different tools:

- Ad Variants
- Sequencing

With ad variants it is possible to define certain elements to be used as part of the display of each ad break. An ad can be display together with an opening/closing clip as well as a bumper ad. Sequencing allows for even more complex settings. By defining a list of break points (that always relate to existing video/streaming content) it is possible to predefine the ad breaks to be displayed during the user session.

Implementation guide

iOS

Contents of the SDK bundle

The smartclip SDK for iOS is shipped in a single zip archive with the following contents:

- SDK as SCMobileSDK-framework-file in the folder SCMobileSDK in three flavors: debug (all architectures) / release (only device targets) and universal (x86-64 as well as device targets)
- API-Docs for the SDK
- Areference implementation of the SDK
- This documentation

Adding the SDK to your project

The smartclip SDK is shipped as iOS framework file. To use the SDK just drag and drop that file into your project. Don't forget to check the "Copy items if needed" checkbox if the framework file is not part of your projects directory.

Open the project settings and select the target that the smartclip SDK is going to be used with. Then open the general tab and add the framework in the Embedded Binaries section.

We recommend to use the debug package for your testing and debugging process and the dedicated release package for your release. However, if your automated process requires all architectures in one release artifact you can use the universal version of it.

After that you need to disable bitcode generation by opening the projects build settings and set "Enable Bitcode" to *NO*.

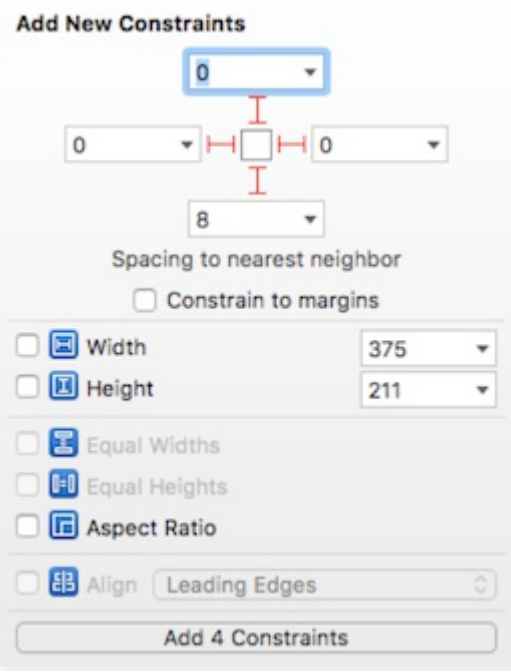
You are now ready to use the smartclip SDK.

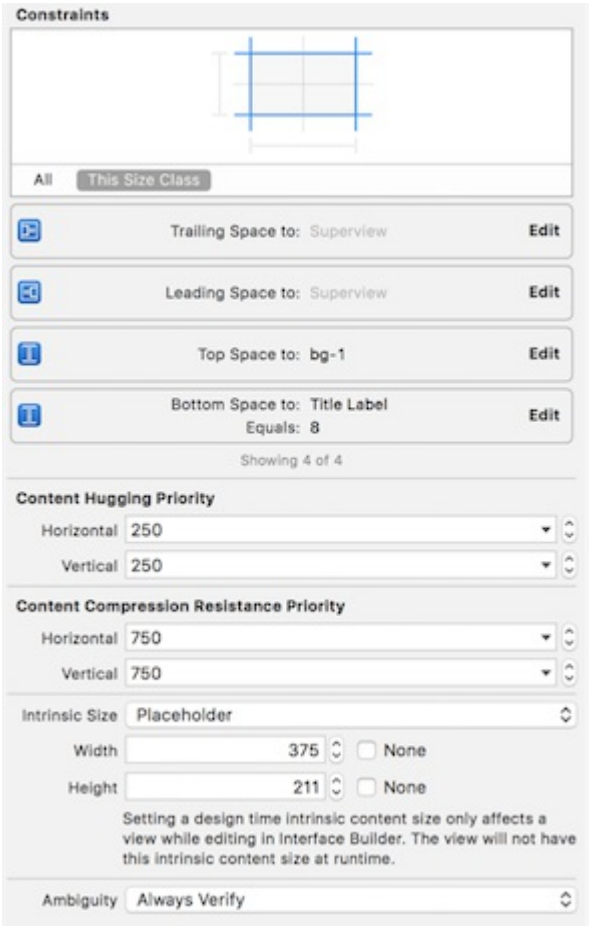
Activate Logs and debug options

For the first steps with the new SDK you might want to activate Debug Logging. To get detailed information you need to set the debug logging with the static variable loggingEnabled to *true*. By default the log level is DEBUG, you can change it: `ScAdLog.setLogLevel(SCAdLog.LogLevel)`

Implementing a basic Out-Stream ad

For the out-stream use case design your app's UI as usual. Create an empty view at the location where the ad should show up. Assign your view four constraints (trailing, leading, top and bottom) and change the intrinsic size from "Default" to "Placeholder". This view will be used as your so called contentView by the *SCAdViewController*. It has to be a subclass of *SCAdContentView* (you can use *AdContentView* from the reference sample code).





In your view controllers *viewDidLoad* method setup the *SCAdViewController* as in the following code example.

```
let config = SCAdConfiguration.defaultConfiguration(with:Constants.testAdUrl)
let controller = SCAdViewController(contentView: anchor, configuration: config)
controller.listener = self
addChildViewController(controller)

class YourViewController: UIViewController, SCAdListener {
  func onEndCallback(controller: SCAdViewController) {
    player?.play()
    controller.removeFromParentViewController()
  }
}
```

Implementing a basic In-Stream ad

For the in-stream use case again start by designing the app's layout like before but use a custom *PlayerView* as anchor instead of UIView. You can take *PlayerView* from the reference sample code. Setup your content video player and smartclip controller in your ViewControllers *viewDidLoad* method.

Setup the video player as following:

```
private func setupAdController() {
  let cfg = SCAdConfiguration.defaultConfiguration(with: Constants.testAdUrl)
```

```
  let adController = SCAdViewController(contentView: videoAnchor, configuration: cfg)
  adController.listener = self
  addChildViewController(adController)
}
```

You now need to listen for events from *SCAdController*. In this particular pre-roll usecase you need to listen for the advertising video to finish. Let your ViewController implement the *SCAdListener* protocol and implement the delegate method *onEndCallback*.

Thats it. When smartclip *SCAdViewController* finished playing it will call *onEndCallback*. The content video player is started then and the smartclip controller and its subsequent views are removed.

Implementing extended configuration options

If you want a more customized configuration as described in the configuration options (for example use IDFAor use extended Click-Through behavior), use the following initializer from the code example. Please note that from SDK version 1.2.5 on you only need to change the values you want to override from their defaults.

```
private func setupAdController() {
  let cfg = SCAdConfiguration.init(headerText: customHeader,
    adURL: requestUrl,
    bundleId: Bundle.main.bundleIdentifier,
    skipOffset: customSkipOffset,
    advertisementId: "YourAdvertisementId",
    behaviorMatrixType: .behaviorMatrixTypeDefault,
    useExtendedClickThrough: true)
  let adController = SCAdViewController(anchor: videoAnchor, configuration: cfg)
  adController.listener = self
  addChildViewController(adController)
}
```

Parameter	Usage	Comment
headerText	Set the "advertising" title displayed in the upper left corner	Default not localized
adURL	The ad tag to use	Mandatory
bundleId	Change the bundle-id used to retrieve the version of the HTML5-component	Debug only
skipOffset	Setting your custom skip offset in seconds	
advertisementId	Your individual user tracking id, like IDFA or custom solutions	Privacy settings and non-trackable rules must be respected
behaviorMatrixType	Change the behavior of your AdView's content	
useExtendedClickThrough	Set this to true if you need to handle clickThrough events individually	see extended clickhandling

Configuration via info.plist

If you want to use a custom CSS with your app, you can do so by adding a reference to the stylesheet to the plist file of your application and store the css-file with your bundle. Just set the value of the Info.plist entry *SCCustomCSSUri* to your custom stylesheet, without the .css file extension. If you don't use one, you can either just leave the entry empty, or just remove the entry entirely from the plist.

Another option that can be triggered via plist is the *SCDisablePlayerScriptUpdate* function, which only uses the JavaScript bundled during the compile process with the App and does not update from server. *Only use this setting after talking to your smartclip contact about how to utilize it.*

Configuring the SCAdEnvironment

There are two possible ways to use the SCAdEnvironment struct when creating the configuration object.

Setting a desired bitrate

The parameter in this scenario is the desired bitrate (plus an optional value for the screenSize - if screenSize is not specified, the native screenSize will be used). The algorithm will try to get as close to the desired bitrate given in selecting from the available media files. It will search downwards first and upwards second.

```
let environment:SCAdEnvironment = SCAdEnvironment.init(desiredBitrate: bitrate, screenSize:
screenSize)
self.adController = SCAdViewController(contentView: adView, configuration:
SCAdConfiguration.defaultConfiguration(with: requestUrl, environment: environment))
```

Defining a deviceType/network manually

Use the deviceType and a value for the current reachability (plus the optional value for screenSize as specified above) to manually override the detected device type and trigger the bitrate scenario for this setup. Please refer to the Jazzy-Documentation for the concrete values to use.

```
let environment = SCAdEnvironment.init(deviceType: deviceType, reachability: reachability,
screenSize: screenSize)
self.adController = SCAdViewController(contentView: adView, configuration:
SCAdConfiguration.defaultConfiguration(with: requestUrl, environment: environment))
```

Listening to events

To get notified about events from your ads you need to register a *SCAdListener* on the *SCAdViewController*. The listener provides the following callbacks:

```
// Listener protocol which is informed about state changes from SCAdSDKController
@objc public protocol SCAdListener:
  class {
    // called when advertisement playback has ended
    // Parameter controller: The parent controller
    @objc optional func onEndCallback(controller: SCAdViewController)

    // called when advertisement playback has started
```

```
    // Parameter controller: The parent controller
    @objc optional func onStartCallback(controller: SCAdViewController)

    // called when an empty video ad was delivered
    // Parameter controller: The parent controller
    @objc optional func onCappedCallback(controller: SCAdViewController)

    // called when prefetching of data for advertisement has finished
    // Parameter controller: The parent controller
    @objc optional func onPrefetchCompleteCallback(controller: SCAdViewController)

    // called on for every ScAdInfo.Type change
    // Parameter controller: The parent controllers
    // Parameter adInfo: current ScAdInfo
    @objc optional func onEventCallback(controller: SCAdViewController, info:
SCAdInfo?)

    // called when the advertisement is clicked
    // link out to landing page can be prevented by catching click events
    //  - Parameter controller: The parent controller
    //  - Parameter targetURL: the advertisement url to be opened
    //  - Returns: true if the given url could be successfully opened
    @objc optional func onClickthru(controller: SCAdViewController, targetURL: URL) -> Bool
  }
```

Handling extended clickThrough

You can activate the extended click thru feature in the following way:

```
var config = SCAdConfiguration.defaultConfiguration(with: Constants.testAdUrl)
config.useExtendedClickThrough = true
```

With this feature activated, you can let the user decide whether he wants to open the url or not, by showing him an alert. The user's decision must then be forwarded to the SDK by calling *clickThruAccepted()* or *clickThruRejected()* on the *SCAdViewControllers* instance.

You can do that the following way:

```
func showClickThruAlert(targetURL: URL) {
  let alert = UIAlertController.init(title: "Open Url",
 message: "Want to open the url?", preferredStyle: UIAlertControllerStyle.alert)
  alert.addAction(UIAlertAction(title: "Yes", style: .default, handler: {
    [weak self] _ in self?.adController?.clickThruAccepted()
    UIApplication.shared.open(targetURL, options: [:], completionHandler: nil)
    }))
  alert.addAction(UIAlertAction(title: "No",
 style: .default,
 handler: {
    [weak self] _ in self?.adController?.clickThruRejected()
  }))
  present(alertController, animated: true, completion: nil)
}
```

Sequencing

The *SCAdSequencingController* allows you to syncronize the ad schedule with a content video. It is necessary to generate a map of sequencing positions as the key pointing to an ad url for this ad block [Position in seconds = Ad url].

```
func setupSequencing() {
```

```
  let sequence = [
    Double(0) : adUrl1,
    Double(10) : adUrl2,

    Double(20) : adUrl3,
    Double(30) : adUrl4
  ]
  sequenceController = SCAdSequencingController( sequenceMap: sequence, anchor: anchorView, duration:
20)
  sequenceController?.sequencingDelegate = self
}
```

The *SequenceController* uses a delegate to communicate with the content video player. To utilize sequencing implement the following methods:

- *playContentVideoPlayer*: play your content video.
- *pauseContentVideoPlayer*: stop your content video.
- *getContentVideoPlayerPosition*: Return the video player position in seconds.

Please note that the SequenceController has to be started manually by calling *start()*. The corresponding call is *stop()*, which will stop the updates.

Android

Contents of the SDK bundle

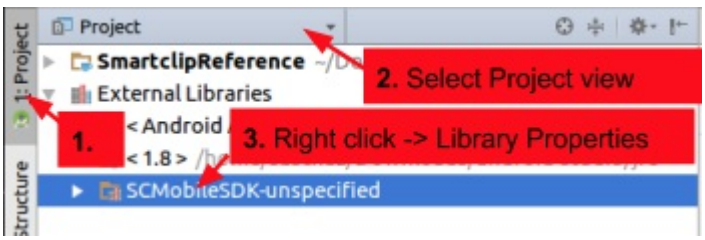
The smartclip SDK for Android is shipped in a single zip archive with the following contents:

- SDK as SCMobileSDK-vx.x.x.aar-file in the folder SCMobileSDK
- API-Docs for the SDK
- Areference implementation of the SDK
- This documentation again referencing the version of the SDK

Adding the SDK to your project

To add this SDK to your project extract the contents of the zip archive to a known location and create a new module in Android Studio and import the aar file (File → New → New Module → Import .JAR/.AAR Package). Now within your apps build.gradle dependency section add the smartclip SDK dependency (replace SCMobileSDK with the module name you defined when importing the .aar file). `dependencies { compile project(':SCMobileSDK') ... }`

After that synchronize the gradle project. To add the smartclip SDK's JavaDoc to the project open the Library Properties of the smartclip SDK:



Press the add button and point the opening dialog to the directory where you placed the smartclip SDK's JavaDoc.

Activate debug logging

To enable debug logging call `ScAdView.DEBUG = true` before calling any other methods of the smartclip SDK.

Implementing a basic Out-Stream ad

For the Out-Stream usecase just place the ScAdView in your layout. It is not recommended to use the ScAdView inside a Recycler- or List-View because the ScAdView cannot handle the proper recreation of the views without sending a new ad request.

```
<de.smartclip.android.ScAdView xmlns:sc="http://schemas.android.com/apk/res-auto"
  android:id="@+id/advertisement"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  sc:adUrl="https://des.smartclip.net/ads?t=de&p=9&pl=testc&test=vast2&sz=400x320"
  sc:headerText="This is the advertisement"/>
```

If not initialized differently the view will use the full screen width. Height will then be set automatically to achieve a 16:9 aspect ratio. Acustom width could be set if needed and the view's height will scale accordingly. With the adUrl attribute you can set your ad tag directly from the XML. The headerText attribute defines a text that is displayed in the top left corner of the ad.

To initialize the ad you can provide the ad URL via XML like shown above or by code in your Activity/Fragment/CustomView with:

```
((ScAdView) findViewById(R.id.advertisement)).setAdURL( 'https://des.smartclip.net/ads?
t=de&p=9372&pl=testc&test=vast2&sz=400x320');
```

Implementing a basic instream ad

To utilize the SDK for an In-Stream usecase, you need to fully cover the video player of your choice with your ScAdView, as in the following example:

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
 xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
android:layout_height="match_parent">
  <FrameLayout
    android:id="@+id/video_container"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintDimensionRatio="H,16:9">
      <com.google.android.exoplayer2.ui.SimpleExoPlayerView
       android:id="@+id/video"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       app:resize_mode="fit"/>
      <de.smartclip.android.ScAdView
       android:id="@+id/advertisement"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"/>
  </FrameLayout>
</android.support.constraint.ConstraintLayout>
```

To initialize the ScAdView just do the same as for the Out-Stream case. Additionally you need to pause and resume the content video according to the ScAdView playstate as shown below.

```
@Override
 protected void onResume() {
  super.onResume();
  // Let this activity listen to ad events.
  adView.addListener(this);
  // To avoid content video and ad playing at the same time
  // only let the content video play when the ad has ended already.
  player.setPlayWhenReady(adView.hasAdEnded());
}
@Override
 protected void onPause() {
  // Always stop the video playback when the activity is paused.
  player.setPlayWhenReady(false);
  // Do not forget to remove the listener you set before.
  adView.removeListener(this);
  super.onPause();
}
@Override
 public void onEndCallback( @NonNull final ScAdView scAdView ) {
  // Start the content video when the ad has ended.
  player.setPlayWhenReady(true);
}
```

Implementing extended configuration options

On Android, custom configuration is achieved by setting the values in calls on the *ScConfigurator* class. When using the *ScAdView* the needed JavaScript SDK is downloaded from a remote location and cached for a maximum of 24 hours. Optionally the JavaScript SDK is also shipped with the smartclip SDK. To use this local version just call before calling any other methods of the smartclip SDK, especially before calling *setAdUrl* or *setVariants*, which trigger communication with the server. `ScSdkCacheConfigurator.disablePlayerScriptUpdate(true)`

If you use the advertiser id macro [IDFAID] you have to provide the advertiser id to the SDK once in the following way: `ScConfigurator.setAdvertiserId("some id")` To activate certain behavior for your app's ad placement the call *setCustomBehaviorMatrix* allows to include a custom set of instructions to control different aspects of the ad. In order to set correct options for your specific usecase get in touch with smartclip's AdTech team.

```
final static String matrix = 'behaviourMatrix : { init: { collapsed: false, paused: false, muted:
false }, onScreen: { muted: false }, offScreen: { paused: false, muted: true }, onClick: { paused:
true, muted: false } }';
ScConfigurator.setCustomBehaviourMatrix(matrix);
```

You can set your custom CSS for advertisements like this:

`ScConfigurator.setCustomStyleSheetUri(Uri.parse('https://some.url.to/stylesheet.css'))`.

The last option available in the *ScConfigurator* is the activation of extended clickthrough. The extended click thru feature can be activated in the following way: `ScConfigurator.useExtendedClickThru(true)`.

Now if the user taps the advertisement an *onClickThru* callback is called and you will receive the corresponding call in *onClickThru*. You are now responsible for opening the landing page and calling *ScAdView.clickThruAccpted()* after that. If your app (or the user) decides not to open the URL be sure to call *ScAdView.clickThruRejected()*.

Setting environment variables via ScConfigurator

In some cases it might be useful to override some environment variables that are used to determine a suitable advertisement media resolution and bitrate. Usually these values are determined automatically from the environment, but there are cases in which your app might have special requirements.

- Device type:
  - Set custom type with *ScConfigurator.setCustomDeviceType(DeviceType)*
  - Available DeviceTypes are: MOBILE, TABLET, DESKTOP and TV
- Network reachability:
  - Set custom type with *ScConfigurator.setCustomNetworkReachability(NetworkReachability)*
  - Available NetworkReachabilities are: WWAN, WIFI and UNREACHABLE
- Display width:
  - Set custom display width in pixel with *ScConfigurator.setCustomDisplayWidth(int)*
  - Setting width smaller 0 will result in using default width
- Display height:
  - Set custom display height in pixel with *ScConfigurator.setCustomDisplayHeight(int)*
  - Setting height smaller 0 will result in using default height
- Desired bitrate:
  - Set desired bitrate in kilobits per second (kbps) with *ScConfigurator.setCustomDesiredBitrate(int)*
  - Setting desired bitrate smaller 0 will result in using default bitrate

Listening to callbacks

To get notified about events from your ads you need to register a *ScListener* on *ScAdView*. The listener provides the following callbacks:

```
public interface ScListener {
  // called when the advertisement has started.
  // @param view associated {@link ScAdView}
  void onStartCallback(@NonNull final ScAdView view);

        // called when the advertisement has finished playback.
  // @param view associated {@link ScAdView}
  void onEndCallback(@NonNull final ScAdView view);

        // called when the user clicks on the advertisement.
  // @param view associated { @link ScAdView }
  // @param targetUrl Url of advertisement that should be opened.
  // @return true to suppress opening of advertisement, otherwise false otherwise.
  // Will be ignored if using { @link ScConfigurator#useExtendedClickThru }
  boolean onClickThru(@NonNull final ScAdView view, @NonNull final String targetUrl);

        // called when the ad was capped. This means that no ad will be
 delivered for the ongoing call.
  // @param view associated { @link ScAdView }
  void onCappedCallback(@NonNull final ScAdView view);
```

```
        // called for every { @link ScAdInfo.Type } change.
  // @param view associated { @link ScAdView }
  // @param adInfo current { @link ScAdInfo }
  void onEventCallback(@NonNull final ScAdView view, @NonNull final ScAdInfo adInfo);
}
```

NOTE: To avoid memory leaks always unregister the listener you registered no later than the Activity/Fragment is destroyed.

Implementing Sequencing

The *ScAdSequencingController* is used to play multiple ads in one single *ScAdView*. It has to be configured with a hashmap containing the position (in seconds) and a corresponding ad-call. Optionally the *ScAdSequencingController* can also be configured to use a specific *ScAdVariants* for a given position (and ad). After calling *start()* the *ScAdSequencingController* is triggered every second. The controller calculate the current position automatically or use the position provided by the *ScSequencePositionSupplier*. Calling *stop()* will stop the update triggering.

```
final HashMap<Long, String> ads = new HashMap<>();
ads.put(5L, "https://des.smartclip.net/ads?t=de&p=9372&pl=testc&test=vast3&sz=400x320");
ads.put(60L, "https://des.smartclip.net/ads?t=de&p=9372&pl=testc&test=vast2&sz=400x320");
sequencingController = new ScAdSequencingController(advertisement, ads, new ScAdVariantsSupplier() {
  @Nullable
  @Override
  public ScAdVariants getVariantForSecond(final long second) {
    return new ScAdVariants("http://some.domain.com/opener",
 "http://some.domain.com/closer",
 "http://some.domain.com/bumper");
  }
},
new ScSequencePositionSupplier());
```