SMARTCLIENTIOS

smartclip

smartclip SDK for native iOS apps

Version history

Document version	Corresponding SDK version	Corresponding JS core	editor
1.0	Native: 2.0.0 (iOS)	Smartclientcore 4.3.0	Karl Szwillus
1.1	Native: 2.1.2 (iOS)	Smartclientcore 4.6.0	Karl Szwillus
1.2	Native: 2.1.2 (iOS)	Smartclientcore 4.6.0	Karl Szwillus
1.3	Native: 2.1.3 (iOS)	Smartclientcore 4.7.0	Karl Szwillus
1.4	Native: 2.1.5 (iOS)	Smartclientcore 4.7.0	Uli Voigt, Karl Szwillus
1.5	Native: 2.2.0 (iOS)	Smartclientcore 4.9.1	Uli Voigt, Karl Szwillus
1.6	Native: 2.3.0 (iOS)	Smartclientcore 4.9.1	Uli Voigt, Karl Szwillus

Latest changes

- 1.6 Bugfix: Timeout to end an adSlot, when the adTag could not be correctly parsed. Added possibility to customize the `UIAlertController` for clickThrough to own needs
- 1.5 Improved visibility checking (instream and outstream) and fullscreen support (instream)
- 1.4 Added information on Playback controls
- 1.3 Overall corrections
- 1.2 Company name change SpotX to smartclip
- 1.1 Improved ad pod Support; new AdError-Object

Contents

- Introduction
- Features
    - AdView-Settings and configuration
    - Providing neccessary information
    - Configuring the environment
    - Event-Callback
    - Retrieving ad and display information
    - Separation clips
    - Sequencing
- Implementation guide

Introduction

The smartclip SDK for iOS is an SDK to display instream and out-stream advertising in native iOS apps. It allows to display various types of video advertisement formats.

Features

Control and display of ads

In the regular instream settings the video player is attached to the SDK to display advertisement. It requires the implementation of an interface, which will serve as connection to the ad server.

It is possible to use one or more ad tags for one ad slot and to configure different types of additional separation clips. In addition a number of additional parameters can be set for the specific environment to pass information to the ad server and to control requested resources.

In an out-stream scenario the SDK also displays video advertisement and offers a number of settings to control appearance and behaviour of the app.

Providing neccessary information

The interface for smartclip SDK allows to control environment and tracking information.

- overwriting Bundle-ID (for debugging and testing purposes)
- supplying reporting/tracking information
- supplying a unique advertising id (IDFA or self-generated)
- adding meta-data on content and context

For a full list of possible information settings please refer to JazzyDoc files in the folder apiDocs.

Setting the IDFA as provided by iOS

The IDFA should be passed over into the SDK if possible. Since the field is not validating, another unique identifier could be generated within the app, which can be used as a means of identification and tracking. The SDK relies on the information passed and does not try to read a value from the system because using IDFA requires special attention during the app submission process to the App Store.

Configuring the environment

- desired bitrate setting

- desired mime-type setting
- information on network/connection stability

Desired bitrate setting

Default bitrate setting of *1000kbps* for ad replay can be overwritten by providing a desired bitrate. The SDK will try to get as close to the desired bitrate as possible from the available media files for each ad tag. If the exact value is not present in the ad response, the SDK is going to search for a lower bitrate media file and only if none is available, the SDK will select a media file of higher bitrate.

Desired mime-type setting

Since support for different video formats varies on certain platforms it is advised to include a list of preferred mime-types to the SDK, which helps the selection process after parsing the ad response. This is relevant when using external components for media playback instead of the *AVPlayer* or similar.

Information on network/connection stability

Configure information on connection stability by using a `NetworkReachability` enum value with
`SCAdEnvironment.reachability = NetworkReachability`
Available values are: `reachabilityWWAN`, `reachabilityWIFI` and `reachabilityUnreachable`.

Event-Callbacks

During an ad slot or ad session the SDK will inform about a number of state information changes that can be accessed by registering a listener/delegate to handle callbacks. This will allow the app to better interact with the advertising content and integrate with the app's control flow.

If there is a registered listener for the `onEventCallback` it will receive all status transitions of the `Adview` or the `Adslot` during playback. Its main purpose is to help reacting to media playback states and possible errors, or for native reporting of different kinds.

In case of an error, information is available in the `getAdError` method. The error itself is triggered by an event that signifies playback, parsing or empty delivery states.

Retrieving ad and display information

Utilizing the `onEventCallback` the application will get information about the current state of the ad player. It starts with the signal of *ON_AD_SLOT_START*, the interpretation of the ad server response, and will iterate through the individual ads that are part of the slot. Callbacks follow the standard VAST events, signalling quartiles and additional information on the type of the ads. Please refer to the apiDocs to get information on all fields and methods available for `SCAdEvent`.

Separation clips

There are some advanced options available to cover more complex ad break scenarios with multiple ads in one single `AdSlot`. Separation clips can be inserted to signify the beginning or end of an ad break, as well as to separate sponsoring ads from regular advertisement.

Sequencing (instream)

The sequencing component allows to create a schedule for a complete content session. It will automatically create and start `AdSlots`, as well as notify the code if any seek or fast-forward/-backward actions are detected. In that case the SDK returns respective ad slots and allows to reset the scheduling. It can be used for various scenarios, even a quite simple preroll-configuration, as it will take care of required monitoring and reporting tasks.

Implementation guide

The SDK supports iOS version 9 and higher.

Contents of the SDK bundle

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

- SDK in three different build targets for universal, production and debug use
- apiDocs
- reference implementation
- this documentation

How to work with the reference implementation

It is recommended to start from the reference implementation, which implements all neccessary interfaces to start a simple project for both instream and out-stream cases.

In an instream setting, where classes will be implemented directly by the app, it is neccessary to implement the interface `SCAdListener` to connect an individual playback solution to the `AdSlots`. Instead of implementing that directly, we recommend to use the sequencer interface for all standard setups, in which case the `SCAdSequencerDelegate` takes the place of the `SCAdListener`.

Adding the smartclip SDK to a project

The smartclip SDK is shipped as an iOS framework file. To use the SDK just drag and drop that file into a project. Activate *Copy items if needed* checkbox if the framework file is not part of the projects directory.

Open project settings and select the target that the smartclip SDK is going to be used with. Under the *General* tab the SDK can be added in the *Embedded Binaries* section.

It is recommended to use the universal package (which is a combined device and simulator version of the SDK) during development for the testing and debugging process. For App Store releases the dedicated release package should be used.

smartclip SDK is now ready for use.

Activate logs and debug options

To get logging information set the debug logging via the static variable `loggingEnabled` to *true*. By default the log level is *Debug*, it can be changed using: `SCAdLog.setLogLevel(Log<Level>)`. Recommended setting for live service is *Error*.

Implementing a basic instream ad with sequencing

For an instream use case start with `SCSequencerViewController` from the reference app package to familiarize with the general setup of ad playback. On iOS (other than on tvOS) smartclip SDK does not manage the player, but requires implementation of listeners and delegates.

Initializing the sequencer:

```
/// tell sequencer about the content video
/// set SCAdConfiguration and a list of SCAdSlot models
private func setupSequencer() {
    /// Create the sequnecer and give it access to the playerController through its facade protocol implementation
    if let facadeDelegate = avPlayerController as SCAdFacadeDelegate? {
        sequencer = SCAdSequencer.init(playerDelegate: facadeDelegate, sessionController: sessionController)

        /// set this viewController as the sequencers delegate
        sequencer?.delegate = self

        sequencer?.contentURL = contentUrl
        sequencer?.configuration = self.configuration()
        sequencer?.adSlots = self.adSlots()

        /// sequencer needs to listen to player events
        avPlayerController?.eventListener = sequencer as? SCPlayerEventListener
        avPlayerController?.avPlayerListener = sequencer as? SCAdListener
    }
}
```

Add the `SCAdSequencerDelegate` which adds functions to restore ad slots to `SCAdListener` protocol, as well as informs about the sequencers state:

```
extension SCSequencerViewController: SCAdSequencerDelegate {
    /// override point for changed presentationSize (called when presentationSize of AVItem changes)
    func presentationSizeChanged(to newSize: CGSize) {
    }

    /// sequncer callback function to start content video
    func playContentVideo(with urlString: String) {
        avPlayerController?.playContentVideo(with: URL.init(string: urlString))
    }

    /// sequncer callback function to pause content video
    func pauseContentVideo() {
        avPlayerController?.pausePlayback()
    }

    /// sequncer callback function to resume content video
    func resumeContentVideo() {
        avPlayerController?.startPlayback()
    }
```

```
    /// slot restoration (example implementation - adjust if required)
    func userDidScrub(_ removedAdSlots: [SCAdSlot], currentRelativePosition:
CGFloat) -> [SCAdSlot] {
        if currentRelativePosition <= 1.0 {
            var reinsertedSlots = Array<SCAdSlot>()
            var index = 1

            for adSlot in removedAdSlots {
                let newValue = (currentRelativePosition + 0.1 * CGFloat(index)) > 1.0 ? 1.0
: (currentRelativePosition + 0.1 * CGFloat(index))

                adSlot.relativeSlotTime = newValue
                reinsertedSlots.append(adSlot)
                index = index + 1
            }

            return reinsertedSlots
        }
        return []
    }

    /// if playback or loading of content video throws an error
    func contentVideoError(_ error: Error) {
        displayAlert(tite: "Content Video Error!",
                     message: String(format: "An error occured while trying to display the
desired content video. \nMessage: %@", error.localizedDescription));
    }

    /// callback method for SCAdEvents
    func onEventCallback(with info:SCAdEvent) {
        switch info.type {
        case ON_AD_SLOT_STARTED:
            self.adSlotStarted()
        case ON_AD_SLOT_FINISHED:
            self.adSlotFinished()
        case ON_AD_SKIPPABLE_STATE_CHANGE:
            self.skipButton.isHidden = false
        case ON_AD_STARTED:
            self.skipButton.isHidden = true
        case ON_AD_PLAYBACK_FINISHED:
            avPlayerController?.getAdInfo { [weak self] (adInfo) in
                if let unwrappedAdInfo = adInfo {
                    if unwrappedAdInfo.variant == "commercial" {
                        if let recentTime = self?.recentClipsTime, let duration =
```

```
self?.avPlayerController?.getDuration() {
                        self?.recentClipsTime = recentTime + duration
                    }
                }
            }
        }
        self.deleteProgressTimer()
    case ON_AD_PAUSED:
        self.deleteProgressTimer()
    case ON_AD_ERROR:
        avPlayerController?.getAdError { (adError) in
            if let unwrappedAdError = adError {
                NSLog("adError: \(unwrappedAdError.errorDescription) code: \
(unwrappedAdError.errorCode)")
            }
        }
    case ON_AD_MUTED:
        self.muteButton.setImage(UIImage(named: "Sound_off"), for: .normal)
    case ON_AD_UNMUTED:
        self.muteButton.setImage(UIImage(named: "Sound_on"), for: .normal)
    default:
        break
    }
}

/// when sequencer has finished sequence
func sequencerFinished() {
    /// Do something, when the sequencer has finished...
}

/// if user clicks on ad and clickThrough is active, sequencer tells its delegate to
open landing page
func displayClickThroughAlert(_ alert: UIAlertController) {
    self.present(alert, animated: true, completion: nil)
}

func adStartsPlayback(with url: URL) {
    if url.absoluteString != opener &&
        url.absoluteString != closer &&
        url.absoluteString != bumper &&
        url.absoluteString != contentUrl {
        createProgressTimer()
    } else {
        deleteProgressTimer()
        progressBar.isHidden = true
```

```
        }
    }
}
```

For the instream use cases the video player is not owned by the SDK, but must be implemented by the user.
However we deliver an example of how it could be implemented, with the `SCAVPlayerController`. There are some things
to be aware of:

The video player talks to the SDK with the help the `SCPlayerEventListener`.
For instance after successful loading of an advertisement video, the video player must call the function `loadAdSuccess()`
on the eventListener, if loading of the video fails, the function `loadAdFailure()` must be called:

```
private func observePlayerItem(change: [NSKeyValueChangeKey : Any]) {
    guard let oldState = change[NSKeyValueChangeKey.oldKey] as? Int,
        let newState = change[NSKeyValueChangeKey.newKey] as? Int else { return }

    switch newState {
    case AVPlayerItem.Status.readyToPlay.rawValue:
        if oldState == AVPlayerItem.Status.unknown.rawValue {
            eventListener?.playerEventCallback(with: EventTypeLoadedData)
            if avPlayer.currentItem == currentAdvertisementItem {
                eventListener?.loadAdSuccess()
                addTimeObserver()
            }
            startPlayback()
        }
    case AVPlayerItem.Status.failed.rawValue:
        if oldState != AVPlayerItem.Status.failed.rawValue {
            if avPlayer.currentItem == currentAdvertisementItem {
                eventListener?.loadAdFailure()
                eventListener?.playerEventCallback(with: EventTypeError)
            } else {
                avPlayerListener?.contentVideoError(avPlayer.currentItem?.error)
            }
        }
    default:
        break
    }
}
```

Also various events must be passed to the eventListener so that the SDK is informed about the current state of the video player.
Please check the `SCAVPlayerController` for details.

Here is a list of those events:

```
typedef enum SCAdPlayerEventType {
    EventTypeLoadedData,
    EventTypePlay,
    EventTypePause,
    EventTypeTimeupdate,
    EventTypeVolumeChanged,
    EventTypeEnded,
    EventTypeError
} SCAdPlayerEventType;
```

Also the videoPlayer controller must implement the `SCAdFacadeDelegate` protocol, through which the SDK tells the video player
when to load an advertisement video and asks the player for different values, like the current time or if the video player is muted.

Implementing an instream ad without sequencing

Using the sequencer is the recommended way of implementing any -- even the simplest -- instream use cases, as it will take care of a lot of reporting and tracking related tasks.

However, it is still possible to implement the `SCAdListener` directly and to implement handling single ad slots.

In `UIViewController` the `SCAdSlotController` can be used together with `SCAdSessionController` and `SCAVPlayerController`. The following code shows neccessary calls to set this up.

```
class SCMobileSDKBaseViewController: UIViewController {

    private let sessionController = SCAdSessionController()

  // ...

  override func viewDidLoad() {
        super.viewDidLoad()
        avPlayerController = SCAVPlayerController.init(with: sessionController)
        adSlotController = sessionController.createAdSlot(withListener:
avPlayerController)
        sessionController.sessionStateListener = self as SCAdSessionStateListener
        avPlayerController.avPlayerListener = self as SCAdListener

        setupAVPlayerViewController()
    }

     override func viewDidAppear(_ animated: Bool) {
        super.viewDidDisappear(animated)
        avPlayerController.eventListener = self.adSlotController as?
SCPlayerEventListener
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        avPlayerController.stopAdSlot()
    }
```

This is not a complete listing of the corresponding class, but a neccessary base setup. Helper methods are needed to create and populate an ad slot, load files and control playback of the player instance. In order to implement `UIViewController` this way, get in touch with smartclip for more detailed sample code.

Implementing a basic Out-stream ad

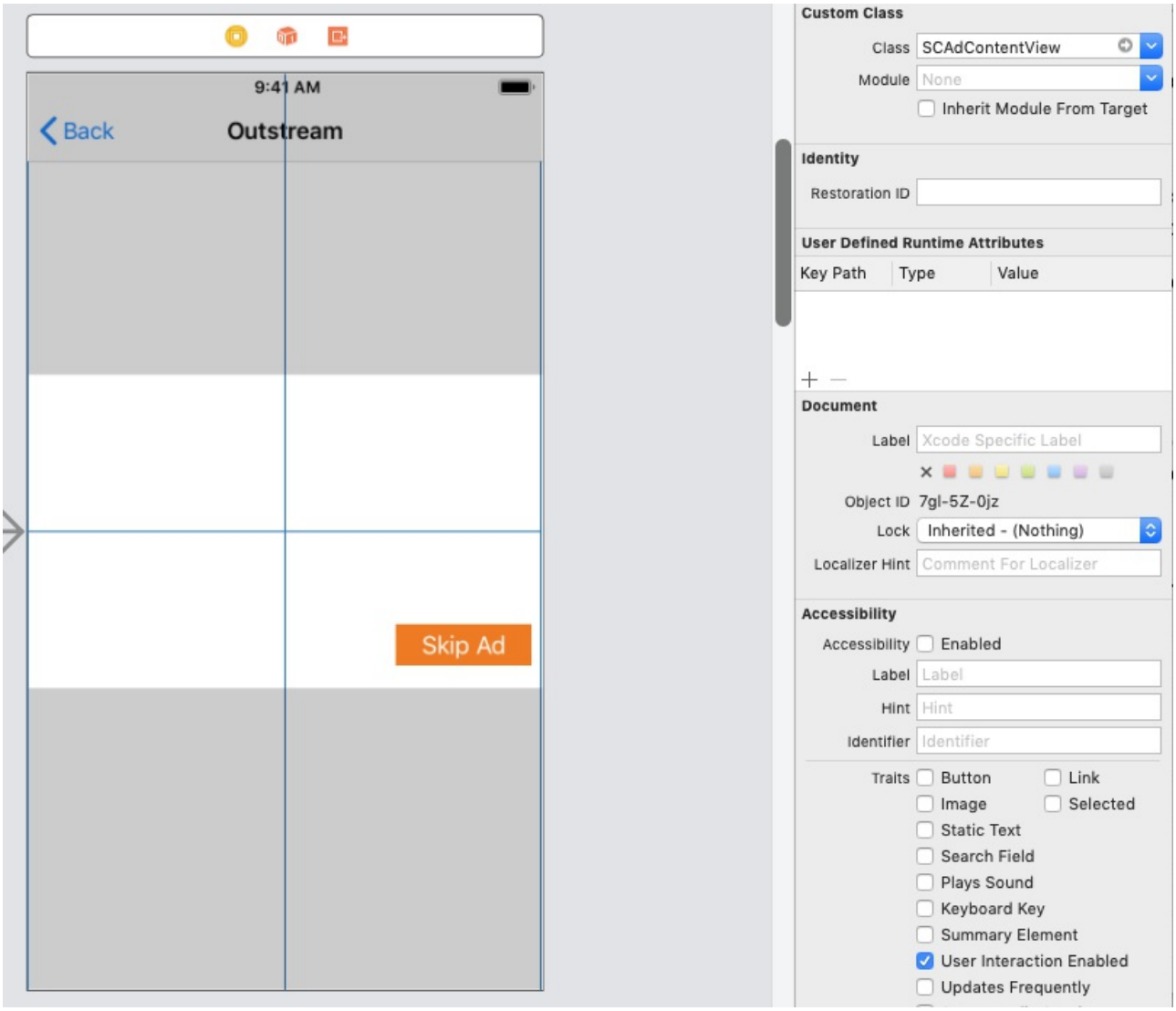
Creating an Out-stream ad playback is similar to the instream ads, when it comes to controlling abstract items like `SCAdSlotController`, but playback is handled by the SDK.

In the reference app there is an example for implementing this scenario, which starts with `SCOutstreamBaseViewController` and which can be used as a basic template for an Out-stream placement.

```
class SCOutstreamBaseViewController: UIViewController {
    @IBOutlet weak var contentView: SCAdContentView!
    var adController: SCAdController?

    override func viewDidLoad() {
        super.viewDidLoad()
        adController = SCAdController.init(sessionStateListener: self as
SCAdSessionStateListener)
        adController?.delegate = self
        adController?.setContentView(self.contentView)
    }
```

It starts with creating the `UIViewController` with an `SCAdContentView` as an outlet to be placed in the storyboard.



Above sections on monitoring by adding `SCAdListener` do apply as well.

Configuring Out-stream placements

To configure Out-stream placements there is a number of options available in the `SCAdConfiguration` object.

```
config.customTitle = "Advertisement"
config.customTitleColor = UIColor.yellow
config.customProgressBarColor = UIColor.yellow
config.skipOffset = 3

config.clickType = ClickableWithConfirmationDialog
```

Please refer to the complete list of options from the JazzyDoc files in the folder apiDocs.

Configuration with *SCAdEnvironment* and *SCAdConfiguration*

There are multiple possible ways to use `SCAdEnvironment` when creating a configuration object. The environment is passed to `SCAdConfiguration`, which is in turn used to configure `SCAdSlotController`.

Setting a desired bitrate and desired mime-types

The SDK will try to play the desired bitrate from the available media files. The SDK will try to get as close to the desired bitrate as possible with the available media files for each ad tag. If the exact value is not present in the ad response, the SDK is going to search for a lower bitrate media file and only if none is available, the SDK will select a media file of higher bitrate.

```
    let environment = SCAdEnvironment.init(macros: macros)
    environment.desiredBitrate = 2000
    environment.desiredMimeTypes = ["video/mp4"]

    let config = SCAdConfiguration.defaultOutstreamConfiguration(with: environment)
    config.adURL = advertisementUrl
    config.variants = SCAdVariants.init(opener: opener, closer: closer, bumper:
bumper)

    adController?.startAdSlot(with: config)
```

Defining a *deviceType*/network manually

Use `deviceType` and a value for `reachability` (plus an optional value for screenSize as specified below) to manually override the detected device type (refer to apiDocs for possible values).

```
  let environment = SCAdEnvironment.init(macros: macros)
  environment.reachability = reachabilityWIFI
  environment.deviceType = deviceTypeTV

  // default screenSize, which is the built in screenSize of the iOS device, can be
overwritten
  // environment.screenSize = CGSize(width: 1000, height: 1000)

  let config = SCAdConfiguration.defaultOutstreamConfiguration(with: environment)
  config.adURL = advertisementUrl
  config.variants = SCAdVariants.init(opener: opener, closer: closer, bumper: bumper)

  adController?.startAdSlot(with: config)
```

Defining desired mime-types

In order to restrict the SDK to certain mime-types, those can be set as environment variable by setting list of desired mime-types to `SCAdEnvironment` as an array.

```
let desiredMimeTypes = ['video/mp4']
environment.desiredMimeTypes = desiredMimeTypes
```

Passing additional information

Additional information is also considered part of the environment and used in the initialization.

```
let macros = SCAdMacros.init()

// add values here
macros.breakPosition = breakPosition
macros.appBundle = Bundle.main.bundleIdentifier!

let environment = SCAdEnvironment.init(macros: macros)
```

The full list of supported information follows (if default is empty the smartclip SDK does not set a value).

Value	Description	
adCategories	Content categories	
appBundle	Bundle ID	
apiFrameworks	API Frameworks	
blockedCategories	blocked content categories	
breakPosition	Position of the AdBreak	Possible values: *SCBreakPositionPreRoll*, *SCBreakP*
clickType	clickType for clickThrough handling	Possible values: *NotClickable*, *ClickableOnFullArea*,
contentId	customer-specific content identifier	
contentPlayhead	CONTENTPLAYHEAD Current time offset	
contentUri	URI of the main media content asset	
domain	DOMAIN	
extensions	Macro: EXTENSION	
ifa	advertising Identifier	
ifaType	rida-Roku id, aaid-Android id, idfa-Apple id	
	list of options indicating attributes of the inventory.	

inventoryState	Possible values: *skippable* to indicate the ad can be skipped, *autoplayed* to indicate the ad is autoplayed with audio unmuted *mautoplayed* to indicate the ad is autoplayed with audio muted *optin* to indicate the user takes an explicit action to knowingly start playback of the ad	
latLong	device position as lat long floats	
limitAdTracking	whether user restricted use of advertising ID	false
mediaMime	array of available MIME types	"video/mp4"
mediaPlayhead	current playhead of content stream	
placementType	one of the *SCPlacementType*	Possible values: *SCPlacementTypeInStream*, *SCPla*
playerCapabilities	list of player capabilities as string list	
regulations	privacy regulations that apply	
verificationVendors	list of verification vendors	
connectionType	SCConnectionType	Possible values: *SCConnectionTypeEthernet*, *SCCo* *SCConnectionTypeCellularUnknownGen*,*SCConnec*
uniqueIdentifier	unique identifier	

Using Opener, Closer and Bumper clips

SCAdVariants specify separation clips that are played back in prominent places within an ad slot. Definition of these separation clips and handling of the video files is done in the source code of the iOS app.

The setting of separation clips is integrated with SCAdConfiguration as these clips are part of the final ad slot play back.

```
let variantObject: SCAdVariants = SCAdVariants(opener: <OpenerURI>, closer: nil,
bumper: nil)
let config = SCAdConfiguration.defaultConfiguration(with: <AdUrl>, variants:
variantObject)

adSlotController.startAdSlot(with: config)
```

Listening to events

To get notified about events from ads SCAdListener must be implemented by the SCAdViewController. The listener provides the following callback.

```
// listener protocol which is informed about state changes from SCAdSDKController
@objc public protocol SCAdListener:
 class {
   // called on for every ScAdInfo.Type change
   // parameter controller: the parent controllers
   // parameter adInfo: current ScAdInfo
   @objc func onEventCallback(with controller: SCAdViewController, info:
SCAdInfo?)
 }
```

The events can be classified by informative events, as well as behavioral events that need to be considered in regards to functioning of the application and user experience.

In error handling an additional call to the SCAdSlotController (or the corresponding control class) has to be made explicitely and will be answered with the SCAdInfo object.

```
case ON_AD_ERROR:
        adSlotController?.getAdError { (adError) in
            if let unwrappedAdError = adError {
                NSLog("adError: \(unwrappedAdError.errorDescription) code: \
(unwrappedAdError.errorCode)")
            }
        }
```

Informative events

These events mainly cover standard VAST events or information on states that have been changed by either player or user.

- ON_AD_SCHEDULED
- ON_AD_PLAYBACK_START
- ON_AD_PLAYING
- ON_AD_FIRST_QUARTILE
- ON_AD_SECOND_QUARTILE
- ON_AD_MID_POINT
- ON_AD_THIRD_QUARTILE
- ON_AD_IMPRESSION
- ON_AD_PLAYBACK_FINISHED
- ON_AD_LINEARITY_CHANGED
- ON_AD_PAUSED
- ON_AD_SKIPPED

Behavioural events

These events may require attention when it comes to displaying or stopping content video. They are expected in the following order.

- ON_AD_MANIFEST_LOADED (ad server response has been parsed and the information on the upcomping slot is commplete)
- ON_AD_SLOT_STARTED (ad slot starts playing)
- ON_AD_STARTED (an ad starts playing)
- ON_AD_FINISHED (an ad finishes playback)
- ON_AD_SLOT_FINISHED (last item has finished playback)
- ON_AD_SLOT_COMPLETE (ad slot came to an end)

Error events

Fired in case of errors

- ON_AD_ERROR

Please also remember to implement the interface method `contentVideoError`. This reports errors of the video player, but is also neccessary to create a complete ad flow.

```
    func contentVideoError(_ error: Error!) {
        displayAlert(tite: "Content Video Error!",
                     message: String(format: "An error occured while trying to display the
desired content video. \nMessage: %@", error.localizedDescription));
    }
```