SMARTCLIENTIOS

smartclip

Migration Guide for Version 3.0.0

Please exchange the old SmarclipSDKiOS.framework with the new SmarclipSDKiOS.xcframework.

After exchanging the old version of the SmarclipSDKiOS with version 3.0.0, you will experience some warnings and errors.
You will have to take the following steps to remove them.

Changes in the SCAVPlayerController implementation:

Changes in existing functions:

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

    switch newState {
    case AVPlayer.TimeControlStatus.waitingToPlayAtSpecifiedRate.rawValue:
        break
    case AVPlayer.TimeControlStatus.paused.rawValue:
        break
    default: //.playing
        // only handle this for changed states!
        if oldState != AVPlayer.TimeControlStatus.playing.rawValue  {

            eventListener?.playerEventCallback(with: EventTypePlay)
        }
    }
}

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 {
                saveLastError()
                eventListener?.loadAdFailure()
                eventListener?.playerEventCallback(with: EventTypeError)
            } else if let item = avPlayer.currentItem, let error = item.error {
```

```
                avPlayerListener?.contentVideoError?(error)
            }
        }
    default:
        break
    }
}

func startPlayback() {
    guard avPlayer.currentItem != nil else { return }

    if avPlayer.timeControlStatus != .playing {
        avPlayer.play()
    }
}

func pausePlayback() {
    guard avPlayer.currentItem != nil else { return }

    if avPlayer.timeControlStatus == .playing {
        avPlayer.pause()
    }
}
```

Please add an observer for the keyPath "presentationSize" of the currentAdvertisementItem (AVPlayerItem):

```
private func observePresentationSize(change: [NSKeyValueChangeKey : Any]) {
    guard let item = avPlayer.currentItem,
    item == currentAdvertisementItem else { return }

    avPlayerListener?.presentationSizeChanged?(to: item.presentationSize)
}
```

Add a member variable:

```
fileprivate var lastError: SCMediaError?`
```

Add a function to save the lastError, depending on the place where the last error occured (either AVPlayer or AVPlayerItem):

```
private func saveLastError() {
    if avPlayer.status == AVPlayer.Status.failed {
        if let error = avPlayer.error {
            lastError = SCMediaError(error: error)
        }
    } else  if  avPlayer.currentItem?.status  ==  AVPlayerItem.Status.failed {
        if let error = avPlayer.currentItem?.error {
            lastError = SCMediaError(error: error)
        }
    }
}
```

Add following functions (according to SCAdPlayerFacade changes):

```
func resumePlayback() {
    startPlayback()
    avPlayerListener?.resumePlayback()
}

func getPlayerSize() -> SCElementSize? {
    return avPlayerListener?.getPlayerSize?()
}

func getViewportSize() -> SCElementSize? {
    return avPlayerListener?.getViewportSize?()
}
```

The SmartclipSDKiOS now supports captions (subtitles). If you want to use them, implement this optional delegate function:

```
func loadAd(withCaptions loadDict: [AnyHashable : Any], disableSeeking: Bool,
completion: SCEmptyCompletionBlock?) -> Bool {
    guard let videoURLString = loadDict["videoURL"] as? String, let subtitleURLString
= loadDict["captionsURL"] as? String else { return false }
    guard let httpSaveVideoURLString = NSString.urlSaveHTTPString(with:
videoURLString),
        let httpSaveSubtitleURLString = NSString.urlSaveHTTPString(with:
subtitleURLString) else { return false }

    guard let videoURL = URL(string: httpSaveVideoURLString) else { return false }
    guard let subtitleURL = URL(string: httpSaveSubtitleURLString) else { return false }

    let localVideoAsset = AVURLAsset(url: videoURL)
```

```
    let subtitleAsset = AVURLAsset(url: subtitleURL)

    guard localVideoAsset.tracks.count > 0, subtitleAsset.tracks.count > 0 else { return
false }

    let videoPlusSubtitles = AVMutableComposition()

    do{
        try? videoPlusSubtitles.insertTimeRange(CMTimeRangeMake(start:
CMTime.zero, duration: localVideoAsset.duration), of: localVideoAsset, at:
CMTime.zero)

        for track in subtitleAsset.tracks {
            let compTrack = videoPlusSubtitles.addMutableTrack(withMediaType:
track.mediaType, preferredTrackID: kCMPersistentTrackID_Invalid)

            try? compTrack?.insertTimeRange(CMTimeRangeMake(start: CMTime.zero,
duration: localVideoAsset.duration),
            of: track,
            at: CMTime.zero)
        }
    }

    if self.currentAdvertisementItem != nil {
        NotificationCenter.default.removeObserver(self, name:
NSNotification.Name.AVPlayerItemDidPlayToEndTime, object:
self.currentAdvertisementItem)
        self.removeAdItemObservers()
    } else if self.contentPlayerItem != nil {
        NotificationCenter.default.removeObserver(self, name:
NSNotification.Name.AVPlayerItemDidPlayToEndTime, object:
self.contentPlayerItem)
    }

    self.currentAdvertisementItem = AVPlayerItem(asset: videoPlusSubtitles)

    NotificationCenter.default.addObserver(self, selector:
#selector(self.adItemFinished), name:
NSNotification.Name.AVPlayerItemDidPlayToEndTime, object:
self.currentAdvertisementItem)
    self.addPlayerObserver()
    self.removeContentItemObserver()
    self.addAdItemObservers()

    DispatchQueue.main.async { [weak self] in
```

```
        self?.avPlayer.replaceCurrentItem(with: self?.currentAdvertisementItem)
        completion?()
    }

    return true
}
```

Rename following functions:

from:

```
func setCurrentContentSource()
```

to:

```
func lockContent()
```

from:

```
func resetCurrentContentSource() -> Bool
```

to:

```
func releaseContent() -> Bool
```

Changes in function declarations (due to nullable/nonnull changes of SCAdFacadeDelegate):

```
func getAdInfo(_ completion: @escaping SCAdInfoCompletionBlock)

func getAdError(_ completion: @escaping SCAdErrorCompletionBlock)

func onEventCallback(_ event: SCAdEvent)

func getError() -> String?

func displayClickThroughAlert(_ alert: UIAlertController)

func loadAd(withUrlString urlString: String, disableSeeking: Bool,
completion:SCEmptyCompletionBlock?) -> Bool
```

Change the implementation of the loadAd(withUrlString urlString: String, disableSeeking: Bool) function in the following way

```
func loadAd(withUrlString urlString: String, disableSeeking: Bool,
completion:SCEmptyCompletionBlock?) -> Bool {
    guard let urlStr = NSString.urlSaveHTTPString(with: urlString),
          let url = URL(string: urlStr) else {

            return false
    }

    if self.currentAdvertisementItem != nil {
        NotificationCenter.default.removeObserver(self, name:
NSNotification.Name.AVPlayerItemDidPlayToEndTime, object:
self.currentAdvertisementItem)
        self.removeAdItemObservers()
    } else if self.contentPlayerItem != nil {
        NotificationCenter.default.removeObserver(self, name:
NSNotification.Name.AVPlayerItemDidPlayToEndTime, object:
self.contentPlayerItem)
    }

    self.currentAdvertisementItem = AVPlayerItem(url: url)
    NotificationCenter.default.addObserver(self, selector:
#selector(self.adItemFinished), name:
NSNotification.Name.AVPlayerItemDidPlayToEndTime, object:
self.currentAdvertisementItem)
    self.addPlayerObserver()
    self.removeContentItemObserver()
    self.addAdItemObservers()

    DispatchQueue.main.async { [weak self] in
        self?.avPlayer.replaceCurrentItem(with: self?.currentAdvertisementItem)
        completion?()
    }

    return true
}
```

Change the implementation of the getError() -> String? function to return your added lastError variable or nil

```
func getError() -> String? {
    guard let lastErrorToReturn = lastError else { return nil }
    lastError = nil
    return lastErrorToReturn.description()
}
```

Changes for Outstream implementation:

Outstream is no longer supported!

Changes if you are using the sequencer:

The SmartclipSDKiOS now is offering a so called "SCSequencedAdController", that handles most of the work, that you had to implement yourself in version 2.x.
You create the SCSequencedAdController in the following way:

```
    let avPlayerController = SCAVPlayerController()
    let adController = SmartclipSDKiOSCreator.createSmartclipSDKForSequencer()
```

Set your SCAdConfiguration, adSlots and contentURL on the adController and set your viewController as the adControllers delegate:

```
    adController.setConfiguration(configuration)
    adController.delegate = self
    adController.setFacadeDelegate(avPlayerController as SCAdFacadeDelegate)
    adController.setAdSlots(adSlots)
    adController.setContentURLString(Constants.contentVideoURL) //whatever
contentVideo url you use
```

Please call the "cleanup" function when you leave your viewController:

```
override func viewDidDisappear(_ animated: Bool) {
    super.viewDidDisappear(animated)
    if !presentingFullscreen {
        adController.cleanup()
    }
}
```

Implement the adControllers delegate functions in your viewController:

```
// MARK: SCAdControllerDelegate functions
extension SCSequencerViewController: SCSequencedAdControllerDelegate {
    func updateProgress(_ progress: CGFloat) {
        progressBar.setProgress(Float(progress), animated: true)
    }

    func configureDisplay(with adViewConfiguration: SCAdViewConfiguration) {
        titleLabel.text = adViewConfiguration.customTitle
        titleLabel.textColor = adViewConfiguration.customTextColor
        progressBar.tintColor = adViewConfiguration.customProgressBarColor
    }

    func displayProgress(_ display: Bool) {
        progressBar.isHidden = !display
    }

    func playerSize() -> CGRect {
        let playerView: UIView = playerViewController.view
        return playerView.bounds
    }

    func viewportSize() -> CGRect {
        return view.bounds
    }

    func displayAlert(_ alert: UIAlertController) {
        if !presentingFullscreen {
            present(alert, animated: true, completion: nil)
        } else {
            fullscreenViewController?.present(alert, animated: true, completion: nil)
        }
    }

    func displayClickThroughAlert() {
        let alertController = UIAlertController(title: "Öffne URL in Browser:", message:
"Willst du das wirklich?", preferredStyle: .alert)

        let negativeAction = UIAlertAction(title: "Nein", style: .default, handler: { [weak
self] _ in
            self?.adController.resumeAdPlayback()
        })

        alertController.addAction(negativeAction)

        let positiveAction = UIAlertAction(title: "Ja", style: .default, handler: { [weak self]
```

```
_ in
            self?.adController.clickThroughURL { (clickThroughURL) in
                guard let clickThroughURL = clickThroughURL, let url = URL(string:
clickThroughURL) else {
                    self?.adController.resumeAdPlayback()
                    return
                }

                UIApplication.shared.open(url, options: [:], completionHandler: nil)
            }
        })

        alertController.addAction(positiveAction)

        self.displayAlert(alertController)
    }

    func adSlotStarted() {
        playerViewController.showsPlaybackControls = false
        titleLabel.isHidden = false

        progressBar.progress = 0
        progressBar.isHidden = false

        clickListener?.isEnabled = true
        skipButton.isHidden = true
        muteButton.isHidden = false

        clickThroughView.isHidden = false

        playerViewController.contentOverlayView?.bringSubviewToFront(titleLabel)

        displayMuted(adController.isMuted())
    }

    func playbackFinished() {
        DispatchQueue.main.asyncAfter(deadline: .now() + 0.4) {
            self.navigationController?.popViewController(animated: true)
        }
    }

    func exitFullscreen(afterPlayback exit: Bool) {
        if #available(iOS 11.0, *) {
            playerViewController.exitsFullScreenWhenPlaybackEnds = exit
        }
```

```
    }

    func adSlotCompleted() {
        playerViewController.showsPlaybackControls = true
        titleLabel.isHidden = true
        progressBar.isHidden = true
        progressBar.setProgress(0, animated: false)

        clickListener?.isEnabled = false
        skipButton.isHidden = true
        muteButton.isHidden = true

        clickThroughView.isHidden = true
    }

    func hideSkipButton(_ hidden: Bool) {
        skipButton.isHidden = hidden
    }

    func displayMuted(_ muted: Bool) {
        let image = muted ? UIImage(named: "Sound_off") : UIImage(named:
"Sound_on")
        muteButton.setImage(image, for: .normal)
    }

    // this function is optional:
    func onEventCallback(with event: SCAdEvent) {
        guard let eventString = event.eventString() else { return }
        print("onEventCallback with event: \(eventString)")

        if event.type == ON_AD_ERROR {
            adController.getAdError { (adError) in
                guard let adError = adError else { return }
                print("ON_AD_ERROR: \(adError.errorDescription) with code: \
(adError.errorCode)")
            }
        }
    }

    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 []
    }
}
```