Instructions to the nexxPlay framework

IMPORTANT:
From v1.2.0 the nexxPlay framework contains the Google IMA sdk. In order to avoid duplicate symbols, please do not add the Google IMA sdk additionally to your project.

1. Adding the framework to an existing project

If you have already added an older version of the nexxPlay framework to your project, please remove the folder "nexxPlay" completely from your project folder. Please also remove the **nexxPlay.framework** and **nexxPlay.bundle** from the project navigator in Xcode.

1. In Xcode, select your project, select your target and open the tab "General"
2. Drag the nexxPlay.framework file from the Finder to the "Embedded Binaries" section.
3. Please make sure that the framework is also included in "Linked Frameworks and Libraries"
4. Change your tab to "Build Phases"
5. Drag the nexxPlay.bundle from the Finder to the "Copy Bundle resources".
6. Still in "Build Phases", click on the "+" icon and add a new run script phase.
7. Copy the code from the file "strip-framework.sh" into the run script phase.
8. **IMPORTANT CHANGE IN XCODE 8**, ONLY Objective-C: Change your tab to "Build Settings" and make sure that **"Always Embed Swift Standard Libraries"** is set to "YES" (the default value is "NO")

2. Simple ViewController examples

Adding a PlayerView

Swift

```
import UIKit
import nexxPlay

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let v_player = PlayerView(frame: CGRect(x: 0, y: 0, width: 300, height: 300))
        view.addSubview(v_player)

       //possible predefinitions
       v_player.overrideMidroll("http://vastURL", interval:5)
       v_player.overrideAutoPlay(true)


        v_player.startPlay("0", client: "571", playmode: "single", param: "40164")
    }
}
```

Objective-C

```
#import "ViewController.h"
#import <nexxPlay/nexxPlay.h>

@interface ViewController ()
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    PlayerView* view = [[PlayerView alloc] initWithFrame:CGRectMake(0.0, 0.0, 300.0,
300.0)];
    [self.view addSubview:view];

    //possible predefinitions
    [v_player overrideMidroll:@"http://vastURL" interval:5];
    [v_player overrideAutoPlay:YES];

    [view startPlay:@"0" client:@"571" playmode:@"single" param:@"40164"
startPosition:0 startItem:0];
}

@end
```

Removing a PlayerView

As soon as the PlayerView is deallocated, the player will automatically stop.

3. Public methods

Defining VAST information prior to the player start

The following methods can be called prior to the initial method startPlay() in order to predefine information for the player. In case the server also provides this information when calling startPlay(), only the information that was not provided before startPlay() is used from the server.

```
public func overridePreroll(url:String)
```

Sets the URL for the VAST preroll.

```
public func overrideMidroll(url:String, interval:Int)
```

Sets the URL and time interval (minutes) for the VAST midroll.

```
public func overrideBanner(url:String, interval:Int)
```

Sets the URL and time interval (minutes) for the VAST banner roll.

```
public func overridePostroll(url:String)
```

Sets the URL for the VAST postroll

```
public func overrideAdProvider(provider:String)
```

Sets the ad provider.

```
public func overrideAdType(adType:String)
```

The different values for ad type can be “vast” and “ima”, “vast” is set as default parameter. In case you want to use the Google IMA SDK to play video ads, override the default setting with “ima”.

```
public func overrideVASTData(provider:String, prerollURL:String,
midrollURL:String, midrollInterval:Int ,bannerURL:String,
bannerInterval:Int,postRollURL:String, adType:String)
```

Sets all VAST information for the player.

```
func overrideMenu(visibility:Int)
```

Updates the setting defining whether the menu is never visible (0), visible on tap (1) or always visible(2)

```
func overrideAutoPlay(autoPlayMode:Bool)
```

Updates the setting defining whether videos should start automatically

```
func overrideExitMode(exitMode:String)
```

Updates the exitMode after the video. The different modes are:

- “replay”: no exit page, only a replay button in the center
- “navigate”: presents an exit page
- “playlist”: nothing is shown

```
func overrideAutoNext(autoNext:Bool)
```

Updates the setting defining whether the next video should start automatically after some seconds

```
func setPlayerBackgroundColor(backgroundColor:String)
```

Updates the background color of the player. The default color is black. Please enter the color as a 6 digit hex value (e.g. white is “ffffff”).

```
func overrideStartMuted(startMuted:Bool)
```

Updates the setting defining whether the player starts muted or not

```
func setDelay(delay:Int)
```

Defines the start position of the video (in seconds).

```
func overrideTitles(visibility:Int)
```

Defines the visibility of the video titles. The titles can be hidden (0), always visible (1) or only visible when the player is fullscreen (2).

```
func addObserverForAllNotifications(observer:AnyObject, selector:String)
```

Convenience function to add an `observer` on all notifications the player will send. Whenever a notification is sent by the player, the function defined by `selector` is called on `observer`. For more information see Notifications (4).

```
func removeObserverForAllNotifications(observer:AnyObject)
```

Convenience function to remove an `observer` on all notifications the player will send.

```
func enableCordova()
```

If the player is used in a cordova app, this method **must** be called before start play to enable cordova specific features.

```
func clearCache()
```

The data received by the nexxPlay API is cached for 30 minutes. Call this method to clear the cache manually.

```
func setDatamode(datamode:String)
```

By default, the video data is received from the API. If your set the data mode to “static”, instead a static url is called to receive all the necessary data.

```
func setGoogleIMAReferenceViewController(viewController:UIViewController)
```

The Google IMA SDK presents the advertisement modally on a UIViewController once the user taps the video ad. If the Google IMA SDK is used but no view controller is set, the advertisement will open in Safari.

```
func setWerbURLRepresentation(url:String)
```

In case any of the VAST urls contains the placeholder “(page.host)”, the given string will replace it.

```
func setViewParentID(mode:String)
```

Sets the parent ID for playlists in order to add references for the reporting.

```
func setPlayLicense(playLicense:Int)
```

Use this method if you want to group videos coming from the same domain ID.

```
func setLoaderSkin(skin:String)
```

Changes the default loading skin. Possible values are:

- “default”: the default loading skin. If you want to use this skin, you don’t need to call this method
- “doublebounce”

```
func setSessionCallReason(reason:String)
```

Use this method if you want to set a reason how the player was opened (for reporting).

```
func setDeviceID(deviceID:String)
```

Use this method if you want to manually set the deviceID for reporting. Otherwise the player will create a deviceID.

```
func setStreamingFilter(filter:String)
```

Use this method if you want to exclude quality tracks from your stream.

```
func setUserHash(userHash:String)
```

Use this method if you have a logged in user with which you want to start your session.

```
func overrideCaptionMode(mode:String)
```

Use this method to override the caption mode for the player. Possible values are:

- none
- select
- selectandstart
- always

```
func disableAds(disablePre:Bool, disableMid:Bool, disableBanner:Bool,
disablePost:Bool)
```

Use this method to override any ad definitions yet set in the player.

```
func setSSL(useSSL:Bool)
```

Use this method to define whether api and media URLs should be created with or without SSL.

Controlling the player

```
func startPlay(sessionID:String, client:String, playmode:String,
param:String, startPosition:Int = 0, startItem:Int = 0)
```

Initial method to get the client data, ad data and video data. The attributes startPosition and startItem are optional.

```
func swap(param:String)
```

This method changes the video ID, thus the player will load the necessary information about the new video and restart with the new data

```
func swapToPos(newParam:String)
```

In case the player currently plays a playlist, it will switch to the video with the provided ID (newParam)

```
func swapComplex(playMode:String, param:String, startPosition:Int = 0,
startItem:Int = 0)
```

Like swap() this method changes the video ID but also lets you define a new playmode as well as the first item to be played (for playlists) and the position, the video is started from (in seconds). The attributes startPosition and startItem are optional.

```
func play()
```

The player starts/continues playing.

```
func pause()
```

The player pauses.

```
func mute()
```

The player is muted.

```
func unmute()
```

The player is unmuted.

```
func seek(time:Int)
```

The current video is set to the defined position (in seconds).

```
func getMediaData(getEnhanced:Bool) -> [String:String]
```

Returns details about the current video as a dictionary. These details contain:

Key	**Value**
mediaID	The id of the current media
hash	The hash value
title	The title of the media
subtitle	The subtitle (optional)
runtime	The runtime of the media (optional)

actors	The actors (optional)
channel	The channel (optional)
channel_id	The channel id (optional)
channel_adref	The channel ad reference (optional)
thumb	The url of the thumb (optional)
uploaded	The UNIX Timestamp stating when the media was uploaded (optional)
created	The UNIX Timestamp stating when the media was created (optional)
orderhint	The order hint (optional)
licenseBy	The order hint (optional)

If there is no current video, the method returns an empty dictionary. If the dictionary is not empty, the attributes mediaID, hash and title are set. All other attributes are optional and may not be available.
If you set the optional `getEnhanced` attribute to `true`, you will get additional parameters:

mediaSession	The session id of the current video
currentDuration	The duration the player is playing. Can be different to currentTime if the user skips seconds or jumps to other timestamps
currentTime	The current timestamp of the playing video
streamType	The stream type (“live” oder “video”)
isAutoPlay	defining whether the video was started automatically (1 or 0)

```
func checkPayment()
```

In case the player was automatically paused because a preview of a video was shown, this method can trigger the check for payment in case the check is positive, the player resumes the video.

4. Notifications

In order to observe the player's behavior the framework provides multiple notifications for the application. Some notifications contain additional data in the `userInfo` dictionary. The notifications are:

nexxPlayErrorNotification
The player shows the error view

nexxPlayStartPlaybackNotification
The player did load the video data

nexxPlayStartPlayNotification
The player starts playing the content video. If there is an ad before the actual content video, this notification is fired once the ad is ended and the content will start

nexxPlayPlayNotification
Whenever the video starts to play (also after pause)

userInfo key	value
byUserAction	Whether the player was started by the user or not

nexxPlayPauseNotification
Whenever the video pauses

userInfo key	value
byUserAction	Whether the player was started by the user or not

nexxPlayEndedNotification
Whenever the video is finished

nexxPlayPlayPosNotification
The player switches to a new video in the playlist.

userInfo key	value
position	The video position

nexxPlayAdCalledNotification
The player asks for ad information

userInfo key	value
mode	The adMode, "ima" or "vast"
type	"preroll", "midroll", "banner" or "postroll"

nexxPlayAdStartedNotification
The player starts a video ad

userInfo key	**value**
mode	The adMode, “ima” or “vast”
type	“preroll”, “midroll”, “banner” or “postroll”

nexxPlayAdEnded Notification
A video ad is finished (or skipped if possible)

userInfo key	**value**
mode	The adMode, “ima” or “vast”
type	“preroll”, “midroll”, “banner” or “postroll”

nexxPlayAdError Notification
An error regarding the video ad occured

userInfo key	**value**
mode	The adMode, “ima” or “vast”
type	“preroll”, “midroll”, “banner” or “postroll”

nexxPlayAdClicked Notification
The video ad was clicked

userInfo key	**value**
mode	The adMode, “ima” or “vast”
type	“preroll”, “midroll”, “banner” or “postroll”

nexxPlayAdResumedNotification
The video ad continues playing after the user clicked on it

userInfo key	**value**
mode	The adMode, “ima” or “vast”
type	“preroll”, “midroll”, “banner” or “postroll”

nexxPlayPreviewStoppedNotification
If the video is shown in preview mode and the preview minutes are over

nexxPlayCordovaCloseNotification
If the player is used in combination with cordova and the user presses the close button (only available when cordova is enabled)

nexxPlayMetaDataLoadedNotification
Every time the meta data for a video is loaded

nexxPlayShowUINotification
Every time the player controls become visible

nexxPlayHideUINotification
Every time the player controls will hide

nexxPlayFullscreenEnteredNotification
Every time the player enters fullscreen

nexxPlayFullscreenExitedNotification
Every time the player exits the fullscreen mode

nexxPlayProgress25Notification, nexxPlayProgress50Notification, nexxPlayProgress75Notification, nexxPlayProgress95Notification
Every time the video duration reaches 25%, 50%, 75% and 95% of its length.

nexxPlayMuteNotification
The player has been muted

nexxPlayUnmuteNotification
The player has been unmuted

nexxPlaySecondNotification
Is sent every second when the player is playing a video

userInfo key	**value**
duration	The played duration

nexxPlayQuarterNotification
Is sent every 15 seconds when the player is playing a video

userInfo key	**value**
progress	The played duration

Observing notifications

The notifications are sent by the NSNotificationCenter. To receive a notification you can use the following code snippet:

Swift

```
NSNotificationCenter.defaultCenter().addObserver(self, selector:
"notificationReceived:", name: nexxPlay.nexxPlayErrorNotification, object: v_player)
```

Objective-C

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(notificationReceived:) name:@"NexxPlayErrorNotification"
object:playerView];
```

The appropriate function that is called when the notification is received:

Swift

```
@objc func notificationReceived(notification:NSNotification) {
        println(notification.name)           // prints “NexxPlayErrorNotification”
        if notification.name == nexxPlay.nexxPlayPlayPosNotification {
            if let userInfo = notification.userInfo {
                if let position = userInfo["position"] as? NSNumber {
                    println(“position: \(position.intValue)”)
                }
            }
        }

}
```

Objective-C

```
- (void)notificationReceived:(NSNotification*)notification {
    NSLog(notification.name);
    if ([notification.name isEqualToString:@"NexxPlayPlayPosNotification"]) {
        if (notification.userInfo != nil) {
            NSNumber* position = [notification.userInfo valueForKey:@"position"];
            if (position != nil) {
                NSLog(@"position: %d", position.intValue);
            }

        }
    }
}
```

- If you have multiple players you can determine the player that sent the notification by checking `notification.object`

If you want to receive all notifications from the player, there is a convenience function:

Swift

```
v_player.addObserverForAllNotifications(observer, selector:”notificationReceived:”)
```

Objective-C

```
[playerView addObserverForAllNotifications:self selector:@"notificationReceived:"];
```

Whenever a notification from `v_player` is received by `observer`, the function `notificationReceived:` is called. With `notification.name` you can determine which notification was received and act recordingly.

5. Fullscreen and orientation behavior

Fullscreen

When the player switches to fullscreen, a subview (containing the video and all controls) of the PlayerView is added to the applications keyWindow (`UIApplication.sharedApplication().keyWindow`). Once the user switches back, the subview is added back to the PlayerView.

Orientation behavior

When the device is rotated, the video automatically rotates accordingly.