AppleTV SDK Integration Guide

AppleTV SDK Integration Guide

This integration guide explains how to integrate the AppNexus AppleTV SDK into your AppleTV application, so you can use AppNexus client-side ad insertion to bid on, track, and measure client-side video advertising within your app. The instructions and examples assume the following:

- You're developing against Apple tvOS 12.0 or higher.
- You have existing instances of AVPlayer and AVPlayerView Controller in your development environment
- You're using the Swift code libraries at https://swift.org/core-libraries/#foundation. The minimum version is SWIFT 4.2.
- You're using Xcode v10.0 or higher.

This application supports the AppNexus `/ptv` and `/vmap` endpoints.

The following sections explain the steps needed to complete the integration, provide a reference for customization options, suggest best practices, and make recommendations for developers who prefer to use Objective-C.

Setting Up the Integration

1. Set Up and Import CocoaPods

CocoaPods is used to distribute the AppNexus SDK and integrate it into your app. Use the following links to install CocoaPods and add it to your project.

- Installing CocoaPods On Your System (cocoapods.org)
- Adding CocoaPods To Your Xcode Project (cocoapods.org)

You can use the following code in your pod spec file:

Sample Podfile

```
target 'MyApp' do
  pod 'AppNexusTVOSSDK', '~> 2.0.0'
end
```

2. Import the `AppNexus AppleTV SDK` module into `UIViewController`

Use the following commands:

```
import ANTVSDK
import AVKit //standard Audio Video Kit library
```

3. Add the `AdControllerProtocol` protocol to `UIViewController`

Use the following commands:

```
class VMAPViewController: UIViewController, AdControllerProtocol {
```

4. Make your instances of `AVPlayerViewController` and `AVPlayer` available

Use the following commands:

```
let appContentVideoPlayer:AVPlayer? = AVPlayer(url: URL(string: "<URL to your content
video>")!)
let appContentViewController:AVPlayerViewController? = AVPlayerViewController()
```

5. Create an instance of AdController

The adController object should be scoped within UIControllerVIew. Use the following command:

```
let adController = AdController()
```

6. (Optional) Define the AdSlot object

We provide the `AdSlot` object as a parameter for communication between application and SDK. `AdSlot` represents an ad break (or "ad pod") that contains multiple VAST objects. As you can see in the example, the `VastData` array accommodates multiple ads in a single break.

Initial "setup" method will return `Array<AdSlot>` representing VMAP ad breaks.

```
@objc public class AdSlot: NSObject, Codable {
    public let breakId: String?;
    public var timeOffset: String?;
    public let breakType: String?;
    public let vastDatas: Array<VastData>;
    public let currentAdId: String?;
    public let timeToShowAdBreak: Int?;
}
```

7. Implement the required "adPlaybackControllerDidSetup" delegate method

Use this delegate method in UIViewController to account for preroll ads before content playback.

```
func adPlaybackControllerDidSetup(adSlots: Array<AdSlot>?) {
        if (adSlots == nil) {
         /// fatal SDK error, start playback of your app content video here
  } else {AdControllerProtocol
   /// SDK called ad URL and got a response from the server and all internal
properties are being set and ready to go
   adController.play();//send play() signal to SDK
  }
    }
```

8. Implement the required `adPlaybackControllerShouldStartAd` delegate method

This delegate method must be implemented in UIViewController to notify SDK you are ready for video ad playback. When the SDK determines that it is time to show a video ad, it will pause the content video and invoke this method with a single "AdSlot" object. The publisher application can then implement any ad-specific behavior. Note that this function should return `true`. If it returns `false`, the ad will not be shown.

```
func adPlaybackControllerShouldStartAd(adSlot: AdSlot?) -> Bool {
  return true // if publisher application allows SDK to play video ad for the "adSlot"
 }
```

9. Implement the required `adPlaybackControllerDidNotifyAdSlotEnded` delegate method

This delegate method must be implemented in UIViewController so the SDK can notify the program that the video ad slot has finished playing all the ad creatives. When this delegate is called, the SDK will resume playback of the content video.

```
func adPlaybackControllerDidNotifyAdSlotEnded(adSlot: AdSlot?) {
  //do publisher application's internal work here
 }
```

10. Implement additional delegate methods

Implementing these delegate methods is required. However, they can remain stub functions.

```
/// a delegate when sdk raised an error
///
/// - Parameters:
///   - adSlot: AdSlot where error occurred
///   - result: ANTVErrorProrotocol
/// - Returns: void
func adPlaybackControllerDidRaiseAnError(adSlot: AdSlot?, result: ANTVErrorProtocol?)
{
    //do stuff on publisher application when SDK reports an error
}

/// A delegate SDK triggered a event
///
/// - Parameters:
///   - adSlot: AdSlot where event triggered
///   - result: any object
/// - Returns: void
func adPlaybackControllerDidNotifyAdSlotEnded(adSlot: AdSlot?) {
    //do stuff on publisher application when SDK notifies an event
}
```

Implementation Methods

Call setup()

Invoke the `setup()` method to initialize the SDK and request an ad. When the SDK is ready, it will invoke the `adPlaybackControllerDidSetup` delegate method. Note that passing `contentVideoPlayerViewController`, `contentVideoPlayer`, and `contentUIViewController` allows the SDK to stop content video playback when it is time to show an ad break, as well as to show ad break marks on the content video timeline.

> To initialize the Apple TV SDK, contentVideoPlayer needs to have video assets at the time when setup() is called.

The following examples show how to invoke setup for VMAP, for an AppNexus placement set, for a VAST URL, and for an AppNexus VAST placement.

```
// VMAP URL
adController.setup(vmapURL:String, contentVideoPlayerViewController:
AVPlayerViewController, contentVideoPlayer: AVPlayer, contentUIViewController:
UIViewController, delegate: AdControllerProtocol)
```

```
// AppNexus Placement Set Id
adController.setup(appNexusPsetId: Int, contentVideoPlayerViewController:
AVPlayerViewController?, contentVideoPlayer: AVPlayer?, contentUIViewController:
UIViewController?, delegate: AdControllerProtocol)
```

```
// VAST URL
adController.setup(vastUrl: String, contentVideoPlayerViewController:
AVPlayerViewController?, contentVideoPlayer: AVPlayer?, contentUIViewController:
UIViewController?,delegate: AdControllerProtocol)
```

```
// AppNexus VAST Placement ID
adController.setup(appNexusMemberId: Int, appNexusPlacementId: Int,
contentVideoPlayerViewController: AVPlayerViewController?, contentVideoPlayer:
AVPlayer?, contentUIViewController: UIViewController?,delegate: AdControllerProtocol
```

Pause/Resume Ad

Invoke `pauseAd()/resumeAd()` to control the video ad. `getPauseStatus()` will return the current pause state of the ad.

```
adController.pauseAd() //pause video ad
adController.resumeAd() //resume video ad
adController.getPauseStatus() //true if video ad paused, false otherwise
```

Skip Ad

To render a skip button, Implement the `adPlaybackControllerDidNotifyAnEvent()` delegate method. The SDK logic adds a target to handle `UIControlEvents.primaryActionTriggered` which will call an API `skip()`. This `VideoEvent.timeToShowSkip` trusts the VAST `skipoffset` attribute.

```
func adPlaybackControllerDidNotifyAnEvent(adSlot: AdSlot?, event: VideoEvent?, data:
String?) {
        if (event == VideoEvent.timeToShowSkip) {

            //render skip button
            button.frame = CGRect(x: 0, y: 0, width: 450, height: 100)
            button.setTitle("Click to Skip", for: .normal)
            button.addTarget(self, action: #selector(self.skipClicked), for:
.primaryActionTriggered)
            self.view.addSubview(button)

            //update focus engine to let user click a button
            self.setNeedsFocusUpdate()
            self.updateFocusIfNeeded()
        }
    }

 @objc func skipClicked() {
        self.adController.skip() //SDK skip call

        //remove target and button
        self.button.removeTarget(nil, action: nil, for: .allEvents)
        self.button.removeFromSuperview()
    }
```

Customizing SDK Behavior

You can use the `setOptions()` method to specify how you want the SDK to look, what text you want to display with ads, whether the video ad displays a skip button, and other features. Note that `setOptions()` must be invoked before `setup()`.

All these settings are optional. The following example shows how to set them:

```
adController.setOptions(
  widthOfAdIndicator: 100,
  leftMarginOfAdIndicator: 200,
  bottomMarginOfAdIndicator: 300,
 backgroundColor: UIColor.black,
  heightOfAdIndicator: 400,
  fontColor: UIColor.blue,
  alphaOfAdIndicator: 0.5,
  widthOfAdCountdown: 500,
  heightOfAdCountdown: 600,
  adText: "advertisement",
  isSkippable: true);
 ...
```

setOptions() Parameter Reference

Parameter	Data Type	Description	Example
`widthOfAdIndicator`	`Int` `? = nil`	The width of the ad indicator in pixels. The ad indicator is the section of the ad indicating it is advertising, and not requested content.	
`leftMarginOfAdIndicator`	`Int` `? = nil`	The width of the left margin of the ad indicator in pixels.	
`bottomMarginOfAdIndicator`	`Int` `? = nil`	The height of the bottom margin of the ad indicator in pixels.	
`backgroundColor`	`UIColor` `? = nil`	The background color of the ad indicator.	
`heightOfAdIndicator`	`Int` `? = nil`	The height of the ad indicator in pixels.	
`fontColor`	`UIColor` `? = nil`	The font color of the ad indicator.	
`alphaOfAdIndicator`	`Double` `? = nil`	The alpha channel (transparency) value of the ad indicator. 1.0=non-transparent. 0.5=50% transparent. 0=completely transparent (and therefore invisible).	
`widthOfAdCountdown`	`Int` `? = nil`	The width of the ad countdown in pixels. The ad countdown is the area that displays the number of seconds the ad plays for. If the ad is skippable, it displays the number of seconds until the viewer can click the skip button.	
`heightOfAdCountdown`	`Int` `? = nil`	The height of the ad countdown in pixels.	

`adText`	`String`
	
	`? = nil`
		The text of the ad indicator (for example Ad)	

`isSkippable`	`Bool`
	
	`? = nil`
		Whether a skip button is displayed, allowing the viewer to skip the ad.	true

`disableSelectGestureOverride`	`Bool`
	
	`? = nil`
		Whether the publisher app, not AppNexus, should control the remote trackpad's gesture action. To let AppNexus override the normal gesture and enable a skip action, set this parameter to `false`.	true

Sample Code for UIViewController

This example uses Swift.

```
import Foundation
import ANTVSDK
import AVKit
import os

class DevViewController: UIViewController,  AdControllerProtocol {

    static let contentURL: String = "<your content video url>"

    let adController = AdController()
    let appContentVideoPlayer:AVPlayer? = AVPlayer(url: URL(string: contentURL)!)
    let appContentViewController:AVPlayerViewController? = AVPlayerViewController()
    let button = UIButton.init(type: .system)

    override func viewDidLoad() {
        super.viewDidLoad()
        self.initANTVSDK()
    }

 deinit {
        adController.dismiss() //dismiss all objects created on ANTVSDK
    }
    func initANTVSDK() -> Void {

  //setup options
        adController.setOptions(isSkippable: true, disableSelectGestureOverride: true)

        adController.setup(vmapURL: "<your vmap url>",
contentVideoPlayerViewController: appContentViewController!, contentVideoPlayer:
appContentVideoPlayer!, contentUIViewController: self, delegate: self)

    }

    func adPlaybackControllerDidSetup(adSlots: Array<AdSlot>?) {
        adController.play()
    }

    func adPlaybackControllerShouldStartAd(adSlot: AdSlot?) -> Bool {
        return true;
    }

    @objc func skipClicked() {
```

```
        self.adController.skip()

        //remove target and button
        self.button.removeTarget(nil, action: nil, for: .allEvents)
        self.button.removeFromSuperview()
    }

    func adPlaybackControllerDidNotifyAdSlotEnded(adSlot: AdSlot?) {
        Logger.debug("got adPlaybackControllerDidNotifyAdSlotEnded")
        exit(0)
    }

    func adPlaybackControllerDidRaiseAnError(adSlot: AdSlot?, result:
ANTVErrorProtocol?) {
        Logger.error("got error from SDK: %@", (result?.description)!)
    }

    func adPlaybackControllerDidNotifyAnEvent(adSlot: AdSlot?, event: VideoEvent?,
data: String?) {
        if (event == VideoEvent.timeToShowSkip) {

            //render skip button
            button.frame = CGRect(x: 0, y: 0, width: 450, height: 100)
            button.setTitle("Click to Skip", for: .normal)
            button.addTarget(self, action: #selector(self.skipClicked), for:
.primaryActionTriggered)
            self.view.addSubview(button)

            //update focus engine to let user click a button
            self.setNeedsFocusUpdate()
            self.updateFocusIfNeeded()
        }
```

```
    }
}
```

Tips and Best Practices

Use the following best practices when implementing the SDK in your app.

Updating the DFP Correlator

When you call a DFP tag with ANTVSDK, the correlator ensures the publisher receives a proper response from the DFP server. Make sure you change the `correlator` parameter on your tag every time you make this call, as shown in the following example:

```
https://pubads.g.doubleclick.net/gampad/ads?sz=640x480&iu=/124319096/external/ad_rule_
samples&ciu_szs=300x250&ad_rule=1&impl=s&gdfp_req=1&env=vp&output=vmap&unviewed_positi
on_start=1&cust_params=deployment%3Ddevsite%26sample_ar%3Dpreonly&cmsid=496&vid=short_
onecue&correlator={unique random number}
```

Letting the AppNexus SDK initiate the content video

Your application should not start the content video on its own: it should wait until the AppNexus SDK has loaded ads, and possibly shown a pre-roll ad. Implementing adPlaybackControllerDidSetup ensures the AppNexus SDK starts your content video. When you use this method, preroll ads will be properly displayed before the content video in your app begins.

Memory management

tvOS ARC(Automatic Reference Counting) checks and cleans most of the used objects in ANTVSDK that are not necessary at the time when the publisher application's UIViewController is dismissed. However, to support and ensure the validity of ARC, calling `adController.dismiss()` on the block of `deinit` block of the publisher applications's UIViewController (as shown in the following example) is required.

```
deinit {
        adController.dismiss() //dismiss all objects created on ANTVSDK
    }
```

Testing in Non-Secure Mode

By default, Xcode does not support non-secure HTTP requests. Certain development-time testing and debugging situations might require using non-secure HTTP tags. To temporarily enable full access to HTTP protocol, check the "Project Info" tab to make sure the project allows arbitrary loads.

Managing x86_64 and ARM64 Binary Versions

A Framework "ANTVSDK.framework" includes x86_64 and ARM64 binary to support both an emulator and an actual Apple TV device. In case you don't want to include the x86_64 binary in your application because of Apple AppStore guidelines, you can simply remove it using `lipo`, as shown in the following example:

```
cd ANTVSDK.framework
lipo -remove x86_64 ANTVSDK -o ANTVSDK
```

You can then check to see if the library only contains the ARM64 binary:

```
<username>:ANTVSDK.framework $ file ANTVSDK
ANTVSDK: Mach-O universal binary with 1 architecture: [arm64: Mach-O 64-bit
dynamically linked shared library arm64]
ANTVSDK (for architecture arm64): Mach-O 64-bit dynamically linked shared library
arm64
<username>:ANTVSDK.framework $
```

Objective-C Project Requirements

As a default, XCode doesn't have a setting to use Swift Standard Libraries for Objective-C projects. For an Objective-C-based publisher application, you'll need to set the "Always Embed Swift Standard Libraries" setting to **Yes** in the Build Settings for the IDE.