AVKit for tvOS
Interactive video playback
Session 506

Dan Wright AVKit Engineer
Video Playback on Apple TV
Video Playback on Apple TV

Siri remote: Touch surface
Video Playback on Apple TV

Siri remote: Touch surface
Siri voice commands
Video Playback on Apple TV

Siri remote: Touch surface
Siri voice commands
Older Apple TV remotes
Video Playback on Apple TV

- Siri remote: Touch surface
- Siri voice commands
- Older Apple TV remotes
- iOS Remote app
Video Playback on Apple TV

- Siri remote: Touch surface
- Siri voice commands
- Older Apple TV remotes
- iOS Remote app
- Bluetooth keyboards
Video Playback on Apple TV

- Siri remote: Touch surface
- Siri voice commands
- Older Apple TV remotes
- iOS Remote app
- Bluetooth keyboards
- Game controllers
Video Playback on Apple TV

- Siri remote: Touch surface
- Siri voice commands
- Older Apple TV remotes
- iOS Remote app
- Bluetooth keyboards
- Game controllers
- Infrared universal remotes
Introducing AVKit for tvOS
Modern playback with a consistent user experience
Scenic Landscapes
5 min 🕒
In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.

First Event  Second Event  Third Event
Siri…

“Go back to the beginning”

“Skip ahead 10 minutes”

“What did she say?”

“Switch to French”
Modern Media Stack
Same on tvOS, iOS, and macOS
Modern Media Stack
Same on tvOS, iOS, and macOS

AVKit
Modern Media Stack
Same on tvOS, iOS, and macOS

AVKit
AVFoundation
CoreMedia
Modern Media Stack

Same on tvOS, iOS, and macOS

- AVKit
- AVFoundation
- UIKit/AppKit
- CoreMedia
Agenda
Agenda

Getting Started with AVKit
Agenda

Getting Started with AVKit
Extending Playback with Features Unique to tvOS
Agenda

Getting Started with AVKit
Extending Playback with Features Unique to tvOS
Best Practices
Getting Started with AVKit
AVPlayerViewController

AVFoundation and AVKit basics

- AVPlayerViewController
  - AVPlayer
  - AVPlayerItem
  - AVAsset
Providing Content
Steps to provide content for AVPlayerViewController
Providing Content

Steps to provide content for AVPlayerViewController

// 1. Create asset from a URL
let asset = AVAsset(url: url)
Providing Content
Steps to provide content for AVPlayerViewController

// 1. Create asset from a URL
let asset = AVAsset(url: url)

// 2. Create a playerItem with the asset
let playerItem = AVPlayerItem(asset: asset)
Providing Content
Steps to provide content for AVPlayerViewController

// 1. Create asset from a URL
let asset = AVAsset(url: url)

// 2. Create a playerItem with the asset
let playerItem = AVPlayerItem(asset: asset)

// 3. Create a player with the playerItem
let player = AVPlayer(playerItem: playerItem)
Providing Content
Steps to provide content for AVPlayerViewController

// 1. Create asset from a URL
let asset = AVAsset(url: url)

// 2. Create a playerItem with the asset
let playerItem = AVPlayerItem(asset: asset)

// 3. Create a player with the playerItem
let player = AVPlayer(playerItem: playerItem)

// 4. Associate player with player view controller
playerViewController.player = player
Providing Content
In one line of code

// All four steps in one line of code.
playerViewController.player = AVPlayer(url: url)
Embedding an Inline Player
For noninteractive playback
Embedding an Inline Player
For noninteractive playback

// 1. Set up playerViewController
let playerViewController = AVPlayerViewController()
playerViewController.player = AVPlayer(url: url)
Embedding an Inline Player
For noninteractive playback

// 1. Set up playerViewController
let playerViewController = AVPlayerViewController()
playerViewController.player = AVPlayer(url: url)

// 2. Set its frame to the inline view (use constraints!)
playerViewController.view.frame = inlineVideoRect
Embedding an Inline Player
For noninteractive playback

// 1. Set up playerViewController
let playerViewController = AVPlayerViewController()
playerViewController.player = AVPlayer(url: url)

// 2. Set its frame to the inline view (use constraints!)
playerViewController.view.frame = inlineVideoRect

// 3. Add it to your view
myViewController.view.addSubview(playerViewController.view)
myViewController.addChildViewController(playerViewController)
Interactive Full-Screen Presentation
For full-user interaction

// If the view was embedded, it will zoom automatically
myViewController.present(playerViewController, animated:true, completion:nil)
Extending the Playback Experience
Advanced features of AVKit on tvOS
Extending the Playback Experience

Introduced in tvOS 9.0
Extending the Playback Experience

Introduced in tvOS 9

Adding noninteractive overlays
Extending the Playback Experience

Introduced in tvOS 9

Adding noninteractive overlays

Restricting playback interaction (requiresLinearPlayback)
Extending the Playback Experience

Introduced in tvOS 9

- Adding noninteractive overlays
- Restricting playback interaction (`requiresLinearPlayback`)
- Providing informational metadata
Extending the Playback Experience

Introduced in tvOS 9

- Adding noninteractive overlays
- Restricting playback interaction \(\text{requiresLinearPlayback}\)
- Providing informational metadata
- Providing navigation markers
Extending the Playback Experience

Introduced in tvOS 9

Adding noninteractive overlays
Restricting playback interaction (requiresLinearPlayback)
Providing informational metadata
Providing navigation markers
Identifying interstitial content
Extending the Playback Experience

New in tvOS 10
Extending the Playback Experience
New in tvOS 10

Changing skipping behavior
Extending the Playback Experience
New in tvOS 10

- Changing skipping behavior
- Presenting content proposals
Overlays

For logos and other overlaid graphics

The playback overlay view lies above the video, but below the controls
Overlays

For logos and other overlaid graphics

The playback overlay view lies above the video, but below the controls
Overlays

For logos and other overlaid graphics

The playback overlay view lies above the video, but below the controls.
Overlays

For logos and other overlaid graphics

The playback overlay view lies above the video, but below the controls.
Views may be static or animated.
Overlays

For logos and other overlaid graphics

The playback overlay view lies above the video, but below the controls.

Views may be static or animated.

Views will not receive focus/events.
Restricting Playback

Restricting playback interaction (requiresLinearPlayback)
User interaction is limited to play/pause
• Fast-forward, scrubbing, and skipping are forbidden
External Metadata

The `externalMetadata` property of `AVPlayerItem` supplements or replaces embedded information:

- Title
- Description
- Genre (Drama, Comedy)
- Media content rating
- Poster artwork
Scenic Landscapes
5 min 🕒
In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.

First Event
Second Event
Third Event
Scenic Landscapes
5 min 📆
In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.
func metadataItem(identifier: String, value: protocol<NSCopying, NSObjectProtocol>?) -> AVMetadataItem? {
    if let actualValue = value {
        let item = AVMutableMetadataItem()
        item.value = actualValue
        item.identifier = identifier
        item.extendedLanguageTag = "und" // undefined (wildcard) language
        return item.copy() as? AVMetadataItem
    }
    return nil
}
func metadataArtworkItem(image: UIImage) -> AVMetadataItem? {
    let item = AVMutableMetadataItem()
    // Choose PNG or JPEG
    item.value = UIImagePNGRepresentation(image)
    item.dataType = kCMMetadataBaseDataType_PNG as String
    item.identifier = AVMetadataCommonIdentifierArtwork
    item.extendedLanguageTag = "und"
    return item.copy() as? AVMetadataItem
}
Creating external artwork items

```swift
var allItems : [AVMetadataItem] = []

allItems.append(metadataItem(identifier: AVMetadataCommonIdentifierTitle, value: "The Title")!)

allItems.append(metadataItem(identifier: AVMetadataCommonIdentifierDescription, value: "Your description goes here.")!)

if let artworkItem = metadataItem(posterImage) {
    allItems.append(artworkItem)
}

allItems.append(metadataItem(identifier: AVMetadataIdentifierQuickTimeMetadataGenre, value: "Comedy")!)

playerItem.externalMetadata = allItems
```
Scenic Landscapes
5 min

In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.
Scenic Landscapes
5 min 🕒
In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.

First Event  Second Event  Third Event
Navigation Marker Groups

For chapters and events

Used for chapters or to identify interesting events

• Events might include things such as scoring or game highlights

Viewers can easily navigate to markers
Navigation Marker Groups

For easy navigation of chapters and events

AVNavigationMarkersGroup

Chapter 1

Chapter 2

Chapter 3
Navigation Marker Groups

For easy navigation of chapters and events

AVNavigationMarkersGroup

Chapter 1

Chapter 2

Chapter 3
Navigation Marker Groups
For easy navigation of chapters and events

Every marker has a title

AVNavigationMarkersGroup

Chapter 1
Chapter 2
Chapter 3
Navigation Marker Groups

For easy navigation of chapters and events

An event group has a title; a chapter group does not
func navigationMarker(title: String, description: String, timeRange: CMTimeRange) -> AVTimedMetadataGroup {
    var items: [AVMetadataItem] = []
    if let titleItem = metadataItem(identifier: AVMetadataCommonIdentifierTitle, value: title) {
        items.append(titleItem)
    }
    if let descriptionItem = metadataItem(identifier: AVMetadataCommonIdentifierDescription, value: description) {
        items.append(descriptionItem)
    }
    return AVTimedMetadataGroup(items: items, timeRange: timeRange)
}
func navigationMarker(title: String, description: String, timeRange: CMTimeRange) -> AVTimedMetadataGroup {
    var items: [AVMetadataItem] = []

    if let titleItem = metadataItem(identifier: AVMetadataCommonIdentifierTitle, value: title) {
        items.append(titleItem)
    }

    if let descriptionItem = metadataItem(identifier: AVMetadataCommonIdentifierDescription, value: description) {
        items.append(descriptionItem)
    }

    return AVTimedMetadataGroup(items: items, timeRange: timeRange)
}
func navigationMarker(title : String, description : String, timeRange : CMTimeRange) -> AVTimedMetadataGroup {
    var items : [AVMetadataItem] = []
    if let titleItem = metadataItem(identifier: AVMetadataCommonIdentifierTitle, value: title) {
        items.append(titleItem)
    }
    if let descriptionItem = metadataItem(identifier: AVMetadataCommonIdentifierDescription, value: description) {
        items.append(descriptionItem)
    }
    return AVTimedMetadataGroup(items: items, timeRange: timeRange)
}
Navigation Marker Groups

Creating navigation markers

```swift
func navigationMarker(title: String, description: String, timeRange: CMTimeRange) -> AVTimedMetadataGroup {
    var items: [AVMetadataItem] = []
    if let titleItem = metadataItem(identifier: AVMetadataCommonIdentifierTitle, value: title) {
        items.append(titleItem)
    }
    if let descriptionItem = metadataItem(identifier: AVMetadataCommonIdentifierDescription, value: description) {
        items.append(descriptionItem)
    }
    return AVTimedMetadataGroup(items: items, timeRange: timeRange)
}
```
Interstitial Content
Collapsing content unrelated to the main video
Interstitial Content
Collapsing content unrelated to the main video

Typically unrelated to the main media
Interstitial Content

Collapsing content unrelated to the main video

Typically unrelated to the main media

An interstitial time range identifies a portion of an asset
Interstition Content

Collapsing content unrelated to the main video

Typically unrelated to the main media

An interstitial time range identifies a portion of an asset

Interstitial time ranges collapse into dots on the transport bar
Interstitial Content

Collapsing content unrelated to the main video

Typically unrelated to the main media

An interstitial time range identifies a portion of an asset

Interstitial time ranges collapse into dots on the transport bar

During scrubbing, interstitial content is hidden
Interstitials

Identifying the asset time ranges

Asset

30 sec

60 sec

9:00

Transport bar

7:30
Identifying the asset time ranges
Interstitial Content

Creating and declaring
Interstitial Content

Creating and declaring

Interstitial content should be stitched into your asset on your server (HLS)
Interstitial Content
Creating and declaring

Interstitial content should be stitched into your asset on your server (HLS)
Declare interstitial time ranges
Interstitial Content
Creating and declaring

Interstitial content should be stitched into your asset on your server (HLS)
Declare interstitial time ranges
Implement delegate methods to enforce playback policy
Declaring interstitial time ranges

```swift
var interstitialTimeRanges = [AVInterstitialTimeRange]()

let start = CMTime(seconds: startInterval, preferredTimescale: 1000)
let duration = CMTime(seconds: durationInterval, preferredTimescale: 1000)

let interstitialTimeRange = AVInterstitialTimeRange(timeRange: CMTimeRange(start: start, duration: duration))

interstitialTimeRanges.append(interstitialTimeRange)

myPlayerItem.interstitialTimeRanges = interstitialTimeRanges
```
func playerViewController(playerViewController: AVPlayerViewController, willPresent interstitial: AVInterstitialTimeRange) {
    // Prevent user navigation inside interstitials
    playerViewController.requiresLinearPlayback = true
}
func playerViewController(playerViewController: AVPlayerViewController, willPresent interstitial: AVInterstitialTimeRange) {
    // Prevent user navigation inside interstitials
    playerViewController.requiresLinearPlayback = true
}

func playerViewController(playerViewController: AVPlayerViewController, didPresent interstitial: AVInterstitialTimeRange) {
    // Allow user navigation outside interstitials
    playerViewController.requiresLinearPlayback = false
}
```swift
func playerViewController(playerViewController: AVPlayerViewController, willPresent interstitial: AVInterstitialTimeRange) {
    // Prevent user navigation inside interstitials
    playerViewController.requiresLinearPlayback = true
}

func playerViewController(playerViewController: AVPlayerViewController, didPresent interstitial: AVInterstitialTimeRange) {
    // Allow user navigation outside interstitials
    playerViewController.requiresLinearPlayback = false
}

func playerViewController(playerViewController: AVPlayerViewController, timeToSeekAfterUserNavigatedFrom oldTime: CMTime, to targetTime: CMTime) -> CMTime {
    // Alter this time to redirect to an interstitial
    let interstitialStartTime =(startTimeOfSkippedInterstice(oldTime, to: targetTime)
    return interstitialStartTime.isValid ? interstitialStartTime : targetTime
}
Skipping Behavior

Modifying the actions associated with skipping

```swift
public enum AVPlayerViewControllerSkippingBehavior : Int {
    case `default`
    case skipItem
}

extension AVPlayerViewController {
    public var skippingBehavior: AVPlayerViewControllerSkippingBehavior
    public var isSkipForwardEnabled: Bool
    public var isSkipBackwardEnabled: Bool
}
```
Skipping Behavior
Modifying the actions associated with skipping

// Skip by-item instead of skip +/- a few seconds
playerViewController.skippingBehavior = .skipItem
playerViewController.isSkipForwardEnabled = true
playerViewController.isSkipBackwardEnabled = true

// Delegate methods respond to skipping by-item
func skipToNextItem(for playerViewController: AVPlayerViewController) {
    let nextPlayerItem = AVPlayerItem(url: nextUrl)
    playerViewController.player?.replaceCurrentItem(nextPlayerItem)
}

func skipToPreviousItem(for playerViewController: AVPlayerViewController) {
    playerViewController.player?.replaceCurrentItem(AVPlayerItem(url: prevUrl))
}
Skipping Behavior
Modifying the actions associated with skipping

// Skip by-item instead of skip +/- a few seconds
playerViewController.skippingBehavior = .skipItem
playerViewController.isSkipForwardEnabled = true
playerViewController.isSkipBackwardEnabled = true

// Delegate methods respond to skipping by-item
func skipToNextItem(for playerViewController: AVPlayerViewController) {
    let nextPlayerItem = AVPlayerItem(url: nextUrl)
    playerViewController.player?.replaceCurrentItem(nextPlayerItem)
}

func skipToPreviousItem(for playerViewController: AVPlayerViewController) {
    playerViewController.player?.replaceCurrentItem(AVPlayerItem(url: prevUrl))
}
Skipping Behavior
Modifying the actions associated with skipping

// Skip by-item instead of skip +/- a few seconds
playerViewController.skippingBehavior = .skipItem
playerViewController.isSkipForwardEnabled = true
playerViewController.isSkipBackwardEnabled = true

// Delegate methods respond to skipping by-item
func skipToNextItem(for playerViewController: AVPlayerViewController) {
    let nextPlayerItem = AVPlayerItem(url: nextUrl)
    playerViewController.player?.replaceCurrentItem(nextPlayerItem)
}

func skipToPreviousItem(for playerViewController: AVPlayerViewController) {
   playerViewController.player?.replaceCurrentItem(AVPlayerItem(url: prevUrl))
}
Skipping Behavior
Modifying the actions associated with skipping

// Skip by-item instead of skip +/- a few seconds
playerViewController.skippingBehavior = .skipItem
playerViewController.isSkipForwardEnabled = true
playerViewController.isSkipBackwardEnabled = true

// Delegate methods respond to skipping by-item
func skipToNextItem(for playerViewController: AVPlayerViewController) {
    let nextPlayerItem = AVPlayerItem(url: nextUrl)
    playerViewController.player?.replaceCurrentItem(nextPlayerItem)
}

func skipToPreviousItem(for playerViewController: AVPlayerViewController) {
    playerViewController.player?.replaceCurrentItem(AVPlayerItem(url: prevUrl))
}
Demo
Basic playback with AVKit

Jonathan Long AVKit Engineer
Content Proposals

Suggesting what to watch next
Episode 2 - More Scenic Beauties

In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.
Episode 2 - More Scenic views

In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.
In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.
In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.
In this episode we will explore beautiful scenery from around the world. We will see breathtaking views of mountains, fields, sunsets, and cities from some spectacular places on this planet.
public class AVContentProposal: NSObject, NSCopying {
    public var contentTimeForTransition: CMTime { get }
    public var automaticAcceptanceInterval: TimeInterval
    public var title: String { get }
    public var previewImage: UIImage? { get }
    public var url: URL?
    public var metadata: [AVMetadataItem]
    public init(contentTimeForTransition: CMTime, title: String, previewImage: UIImage?)
}

extension AVPlayerItem {
    public var nextContentProposal: AVContentProposal?
}
public class AVContentProposal : NSObject, NSCopying {

    public var contentTimeForTransition: CMTime { get }
    public var automaticAcceptanceInterval: TimeInterval
    public var title: String { get }
    public var previewImage: UIImage? { get }
    public var url: URL?
    public var metadata: [AVMetadataItem]

    public init(contentTimeForTransition: CMTime, title: String, previewImage: UIImage?)
    }

extension AVPlayerItem {
    public var nextContentProposal: AVContentProposal?
    }
public class AVContentProposal : NSObject, NSCopying {
    public var contentTimeForTransition: CMTime { get }
    public var automaticAcceptanceInterval: TimeInterval
    public var title: String { get }
    public var previewImage: UIImage? { get }
    public var url: URL?
    public var metadata: [AVMetadataItem]
    public init(contentTimeForTransition: CMTime, title: String, previewImage: UIImage?)
}

extension AVPlayerItem {
    public var nextContentProposal: AVContentProposal?
}
public class AVContentProposal : NSObject, NSCopying {
    public var contentTimeForTransition: CMTime { get }
    public var automaticAcceptanceInterval: TimeInterval
    
    public var title: String { get }
    public var previewImage: UIImage? { get }
    public var url: URL?
    public var metadata: [AVMetadataItem]

    public init(contentTimeForTransition: CMTime, title: String, previewImage: UIImage?)
}

extension AVPlayerItem {
    public var nextContentProposal: AVContentProposal?
}
Content Proposals
Suggesting what to watch next

```swift
public class AVContentProposal : NSObject, NSCopying {
    public var contentTimeForTransition: CMTime { get }
    public var automaticAcceptanceInterval: TimeInterval
    public var title: String { get }
    public var previewImage: UIImage? { get }
    public var url: URL?
    public var metadata: [AVMetadataItem]
    public init(contentTimeForTransition: CMTime, title: String, previewImage: UIImage?)
}

extension AVPlayerItem {
    public var nextContentProposal: AVContentProposal?
}
```
Content Proposals

Creating the content proposal

// Create the proposal
Content Proposals
Creating the content proposal

// Create the proposal
let contentProposal = AVContentProposal(contentTimeForTransition: kCMTimeZero, title: "Happy Hijinks S9 E2", previewImage: previewImage)
// Create the proposal
let contentProposal = AVContentProposal(contentTimeForTransition: kCMTimeZero, title: "Happy Hijinks S9 E2", previewImage: previewImage)
contentProposal.url = urlForHappyHijinksS9E2
// Create the proposal
let contentProposal = AVContentProposal(contentTimeForTransition: kCMTimeZero, title: "Happy Hijinks S9 E2", previewImage: previewImage)
contentProposal.url = urlForHappyHijinksS9E2

// Now assign the proposal to the player item which should show the proposal
playerItemForHappyHijinksS9E1.nextContentProposal = contentProposal
public protocol AVContentProposalDelegate {

    optional public func playerViewController(playerViewController: AVPlayerViewController, shouldPresent proposal: AVContentProposal) -> Bool

    optional public func playerViewController(playerViewController: AVPlayerViewController, didAccept proposal: AVContentProposal)

    optional public func playerViewController(playerViewController: AVPlayerViewController, didReject proposal: AVContentProposal)

}
public protocol AVContentProposalDelegate {

    optional public func playerViewController(playerViewController: AVPlayerViewController, shouldPresent proposal: AVContentProposal) -> Bool

    optional public func playerViewController(playerViewController: AVPlayerViewController, didAccept proposal: AVContentProposal)

    optional public func playerViewController(playerViewController: AVPlayerViewController, didReject proposal: AVContentProposal)

}
public protocol AVContentProposalDelegate {

    optional public func playerViewController(playerViewController: AVPlayerViewController, shouldPresent proposal: AVContentProposal) -> Bool

    optional public func playerViewController(playerViewController: AVPlayerViewController, didAccept proposal: AVContentProposal)

    optional public func playerViewController(playerViewController: AVPlayerViewController, didReject proposal: AVContentProposal)

}
public protocol AVContentProposalDelegate {

    optional public func playerViewController(playerViewController: AVPlayerViewController, shouldPresent proposal: AVContentProposal) -> Bool

    optional public func playerViewController(playerViewController: AVPlayerViewController, didAccept proposal: AVContentProposal)

    optional public func playerViewController(playerViewController: AVPlayerViewController, didReject proposal: AVContentProposal)

}
public class AVContentProposalViewController : UIViewController {
    public var contentProposal: AVContentProposal? { get }
    weak public var playerViewController: AVPlayerViewController? { get }
    public var preferredPlayerViewFrame: CGRect { get }
    public var playerLayoutGuide: UILayoutGuide { get }
    public var dateOfAutomaticAcceptance: Date?
    public func dismissContentProposal(for action: AVContentProposalAction, animated: Bool, completion block: (() -> Void)? = nil)
}
public class AVContentProposalViewController : UIViewController {
    public var contentProposal: AVContentProposal? { get }
    weak public var playerViewController: AVPlayerViewController? { get }
    public var preferredPlayerViewFrame: CGRect { get }
    public var playerLayoutGuide: UILayoutGuide { get }
    public var dateOfAutomaticAcceptance: Date?
    public func dismissContentProposal(for action: AVContentProposalAction, animated: Bool, completion block: (() -> Void)? = nil)

public class AVContentProposalViewController : UIViewController {
    public var contentProposal: AVContentProposal? { get }
    weak public var playerViewController: AVPlayerViewController? { get }
    public var preferredPlayerViewFrame: CGRect { get }
    public var playerLayoutGuide: UILayoutGuide { get }
    public var dateOfAutomaticAcceptance: Date?
    
    public func dismissContentProposal(for action: AVContentProposalAction, animated: Bool, completion block: (() -> Void)? = nil)
    {
    }
}
Content Proposals

Responding to delegate notifications

```swift
func playerViewController(playerViewController: AVPlayerViewController, shouldPresent proposal: AVContentProposal) -> Bool {

    // Set up a custom presentation just-in-time
    playerViewController.contentProposalViewController = MyContentProposalViewController()
    return true
}
```
func playerViewController(playerViewController: AVPlayerViewController, 
didAccept proposal: AVContentProposal) {
    // Replace the current AVPlayerItem with the proposed content
    guard let player = playerViewController.player, let url = proposal.url
    else { return }
    player.replaceCurrentItem(AVPlayerItem(url: url))
}
Demo

Content Proposals

Jonathan Long AVKit Engineer
Best Practices
with AVKit on tvOS
Best Practices
With AVKit on tvOS
Best Practices

With AVKit on tvOS

Let `present` handle zooming from inline player views
Best Practices
With AVKit on tvOS

Let `present` handle zooming from inline player views
Playback is only interactive when full-screen
Best Practices
With AVKit on tvOS

Let `present` handle zooming from inline player views
Playback is only interactive when full-screen
Use the new content proposal API
Best Practices
With AVKit on tvOS

Let `present` handle zooming from inline player views
Playback is only interactive when full-screen
Use the new content proposal API
Observe the player/player item `error` property
Best Practices
With AVKit on tvOS

Avoid toggling `showsPlaybackControls`
Avoid toggling `showsPlaybackControls`

Avoid adding supplemental gestures to playback
Best Practices
With AVKit on tvOS

Avoid toggling `showsPlaybackControls`

Avoid adding supplemental gestures to playback

Do not overload the Select button or touch surface gestures
Best Practices
With AVKit on tvOS

Replace your asset upon `AVErrorMediaServicesWereReset`
Best Practices
With AVKit on tvOS

Replace your asset upon AVErrorMediaServicesWereReset

Other sessions with best practices for playback:

<table>
<thead>
<tr>
<th>Advances in AVFoundation Playback</th>
<th>Mission</th>
<th>Wednesday 9:00AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mastering Modern Media Playback</td>
<td></td>
<td>WWDC 2014</td>
</tr>
</tbody>
</table>
Summary
Summary

Standard playback controls and behaviors
Summary

Standard playback controls and behaviors
Support for remotes, game controllers, and Siri
Summary

Standard playback controls and behaviors
Support for remotes, game controllers, and Siri
Full access to media stack
Summary

Standard playback controls and behaviors
Support for remotes, game controllers, and Siri
Full access to media stack
Powerful new APIs
Summary

Standard playback controls and behaviors
Support for remotes, game controllers, and Siri
Full access to media stack
Powerful new APIs
Easy to get started
More Information

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advances in AVFoundation Playback</td>
<td>Mission</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>What’s New in HTTP Live Streaming</td>
<td>Mission</td>
<td>Wednesday 3:00PM</td>
</tr>
<tr>
<td>Developing tvOS Apps Using TVMLKit: Part 1</td>
<td>Mission</td>
<td>Wednesday 1:40PM</td>
</tr>
<tr>
<td>Developing tvOS Apps Using TVMLKit: Part 2</td>
<td>Nob Hill</td>
<td>Thursday 4:00PM</td>
</tr>
<tr>
<td>Mastering Modern Media Playback</td>
<td></td>
<td>WWDC 2014</td>
</tr>
</tbody>
</table>