Reaching the Big Screen with AirPlay 2

Jonathan Bennett, Connected Media
Marty Pye, AVKit
AirPlay Overview

Watch

Listen

Share
AirPlay Overview

Watch

Listen

Share
Works with Apple AirPlay
4K HDR
Best quality video
Best quality video
AirPlay picker
Best quality video
AirPlay picker
Remote controls
Best quality video
AirPlay picker
Remote controls
AirPlay multitasking
Best quality video
AirPlay picker
Remote controls
AirPlay multitasking
Long-form video apps
Best quality video
AirPlay picker
Remote controls
AirPlay multitasking
Long-form video apps
Best Quality Video

HTTP Live Streaming
• Include 4K + HDR + surround sound variants
• Full range of variants per codec
Best Quality Video

HTTP Live Streaming
• Include 4K + HDR + surround sound variants
• Full range of variants per codec
• Provide I-frame variants for smooth seeking
Best Quality Video

HTTP Live Streaming
• Include 4K + HDR + surround sound variants
• Full range of variants per codec
• Provide I-frame variants for smooth seeking

FairPlay Streaming
Smooth Playback

HTTP Live Streaming **EXT-X-DISCONTINUITY** tag

- Avoid switching video formats
Smooth Playback

HTTP Live Streaming **EXT-X-DISCONTINUITY** tag

- Avoid switching video formats

AVQueuePlayer
Smooth Playback

HTTP Live Streaming EXT-X-DISCONTINUITY tag
• Avoid switching video formats

AVQueuePlayer

AVPlayer.replaceCurrentItem(with item:)
Best quality video
AirPlay picker
Remote controls
AirPlay multitasking
Long-form video apps
AirPlay Picker

Add a picker to your app

Decide how to order AirPlay destinations
Recommended API

AVRoutePickerView
Available iOS 11 / macOS 10.15
Recommended API

AVRoutePickerController
Available iOS 11 / macOS 10.15

MPVolumeView
Route Button properties deprecated
Add a Picker

let routePickerView = AVRoutePickerView()
Prioritize Video Devices

```swift
let routePickerView = AVRoutePickerView()
routePickerView.prioritizesVideoDevices = true
```
Prioritize Video Devices
Prioritize Video Devices
Best quality video
AirPlay picker
Remote controls
AirPlay multitasking
Long-form video apps
AirPlay
This video is playing on “Living Room”
Becoming the Now Playing App

Support remote controls

Initiate playback
Becoming the Now Playing App

Support remote controls

Initiate playback

• **iOS:** Activate with a non-mixable `AVAudioSession category`
  - Example: `.playback category`
Becoming the Now Playing App

Support remote controls

Initiate playback

- **iOS**: Activate with a non-mixable `AVAudioSession` category
  - Example: `.playback` category
- **macOS**: Set `MPNowPlayingInfoCenter.playbackState`
Supporting Remote Controls

AVPlayerViewController handles remote controls
Supporting Remote Controls

AVPlayerViewController handles remote controls

MPRemoteCommandCenter and MPNowPlayingInfoCenter
Register for Remote Controls
Register for Remote Controls
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()

    // Add handler for play command
    commandCenter.playCommand.addTarget { [unowned self] event in
        if self.player.rate == 0.0 {
            self.player.play()
            return .success
        }
        return .commandFailed
    }

    // Add handler for pause & toggle playback Commands
    commandCenter.pauseCommand.addTarget { /* ... */ }
    commandCenter.togglePlayPauseCommand.addTarget { /* ... */ }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()

    // Add handler for play command
    commandCenter.playCommand.addTarget { [unowned self] event in
        if self.player.rate == 0.0 {
            self.player.play()
            return .success
        }
        return .commandFailed
    }

    // Add handler for pause & toggle playback Commands
    commandCenter.pauseCommand.addTarget { /* ... */ }
    commandCenter.togglePlayPauseCommand.addTarget { /* ... */ }
}
```swift
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()

    // Add handler for play command
    commandCenter.playCommand.addTarget { [unowned self] event in
        if self.player.rate == 0.0 {
            self.player.play()
            return .success
        }
        return .commandFailed
    }

    // Add handler for pause & toggle playback Commands
    commandCenter.pauseCommand.addTarget { /* ... */ }
    commandCenter.togglePlayPauseCommand.addTarget { /* ... */ }
}
```
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()

    // Add handler for play command
    commandCenter.playCommand.addTarget {
        unowned self in
        if self.player.rate == 0.0 {
            self.player.play()
            return .success
        }
        return .commandFailed
    }

    // Add handler for pause & toggle playback Commands
    commandCenter.pauseCommand.addTarget { /* ... */ }
    commandCenter.togglePlayPauseCommand.addTarget { /* ... */ }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()

    // Add handler for play command
    commandCenter.playCommand.addTarget { [unowned self] event in
        if self.player.rate == 0.0 {
            self.player.play()
        }
        return .success
    }
    return .commandFailed
}

// Add handler for pause & toggle playback Commands
commandCenter.pauseCommand.addTarget { /* ... */ }
commandCenter.togglePlayPauseCommand.addTarget { /* ... */ }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()

    // Add handler for play command
    commandCenter.playCommand.addTarget {
        unowned self in
        if self.player.rate == 0.0 {
            self.player.play()
            return .success
        }
        return .commandFailed
    }

    // Add handler for pause & toggle playback Commands
    commandCenter.pauseCommand.addTarget {
        /* ... */
    }
    commandCenter.togglePlayPauseCommand.addTarget {
        /* ... */
    }
}
Provide Now Playing Media Info
Provide Now Playing Media Info
Provide Now Playing Media Info

Item metadata

Playback state
Provide Now Playing Media Info

- Item metadata
- Playback state
Provide Now Playing Media Info

- Item metadata
- Playback state
// Update all Now Playing info when AVPlayerItem changes
func handleItemChanged(item: AVPlayerItem, metadata: NowPlaybleStaticMetadata) {
    // Set static info like title, artist, etc
    setNowPlayingMetadata(metadata)
    // Set playback metadata, if available
    if let playbackMetadata = playbackMetadata(for: item) {
        setNowPlayingPlaybackInfo(playbackMetadata)
    }
}

// Update playback state Now Playing info for significant events like rate changes and seeks
func handlePlaybackStateChanged(metadata: NowPlaybleDynamicMetadata) {
}

func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
  // Set static info like title, artist, etc
  setNowPlayingMetadata(metadata)
  // Set playback metadata, if available
  if let playbackMetadata = playbackMetadata(for: item) {
    setNowPlayingPlaybackInfo(playbackMetadata)
  }
}

func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
  setNowPlayingPlaybackInfo(metadata)
}
func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
    // Set static info like title, artist, etc
    setNowPlayingMetadata(metadata)
    // Set playback metadata, if available
    if let playbackMetadata = playbackMetadata(for: item) {
        setNowPlayingPlaybackInfo(playbackMetadata)
    }
}

func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
    // Update playback state Now Playing info for significant events like rate changes and seeks
    setNowPlayingPlaybackInfo(metadata)
}
func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
    // Update static info like title, artist, etc
    updateNowPlayingMetadata(metadata)

    // Update playback metadata, if available
    if let playbackMetadata = playbackMetadata(for: item) {
        updateNowPlayingPlaybackInfo(playbackMetadata)
    }
}

func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
    updateNowPlayingPlaybackInfo(metadata)
}
// Update all Now Playing info when AVPlayerItem changes
func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
    // Update static info like title, artist, etc
    updateNowPlayingMetadata(metadata)

    // Update playback metadata, if available
    if let playbackMetadata = playbackMetadata(for: item) {
        updateNowPlayingPlaybackInfo(playbackMetadata)
    }
}

// Update playback state Now Playing info for significant events like rate changes and seeks
func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
    updateNowPlayingPlaybackInfo(metadata)
}
// Update all Now Playing info when AVPlayerItem changes
func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
    // Update static info like title, artist, etc
    updateNowPlayingMetadata(metadata)
    // Update playback metadata, if available
    if let playbackMetadata = playbackMetadata(for: item) {
        updateNowPlayingPlaybackInfo(playbackMetadata)
    }
}

// Update playback state Now Playing info for significant events like rate changes and seeks
func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
    updateNowPlayingPlaybackInfo(metadata)
}
func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
  // Update static info like title, artist, etc
  updateNowPlayingMetadata(metadata)

  // Update playback metadata, if available
  if let playbackMetadata = playbackMetadata(for: item) {
    updateNowPlayingPlaybackInfo(playbackMetadata)
  }
}

func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
  updateNowPlayingPlaybackInfo(metadata)
}
// Update all Now Playing info When AVPlayerItem changes
func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
    // Update static info like title, artist, etc
    updateNowPlayingMetadata(metadata)

    // Update playback metadata, if available
    if let playbackMetadata = playbackMetadata(for: item) {
        updateNowPlayingPlaybackInfo(playbackMetadata)
    }
}

// Update playback state Now Playing info for significant events like rate changes and seeks
func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
    updateNowPlayingPlaybackInfo(metadata)
}
func updateNowPlayingMetadata(_ metadata: NowPlayableStaticMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    nowPlayingInfo[MPMediaItemPropertyTitle] = metadata.title
    nowPlayingInfo[MPMediaItemPropertyArtwork] = metadata.artwork

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
```swift
func updateNowPlayingMetadata(_ metadata: NowPlayableStaticMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    nowPlayingInfo[MPMediaItemPropertyTitle] = metadata.title
    nowPlayingInfo[MPMediaItemPropertyArtwork] = metadata.artwork

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
```
func updateNowPlayingMetadata(_ metadata: NowPlayableStaticMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    nowPlayingInfo[MPMediaItemPropertyTitle] = metadata.title
    nowPlayingInfo[MPMediaItemPropertyArtwork] = metadata.artwork

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
func updateNowPlayingMetadata(_ metadata: NowPlayableStaticMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    nowPlayingInfo[MPMediaItemPropertyTitle] = metadata.title
    nowPlayingInfo[MPMediaItemPropertyArtwork] = metadata.artwork

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
// Update all Now Playing info when AVPlayerItem changes
func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
    // Update static info like title, artist, etc
    updateNowPlayingMetadata(metadata)

    // Update playback metadata, if available
    if let playbackMetadata = playbackMetadata(for: item) {
        updateNowPlayingPlaybackInfo(playbackMetadata)
    }
}

// Update playback state Now Playing info for significant events like rate changes and seeks
func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
    updateNowPlayingPlaybackInfo(metadata)
}
// Update all Now Playing info When AVPlayerItem changes

func handleItemChanged(item: AVPlayerItem, metadata: NowPlayableStaticMetadata) {
    // Update static info like title, artist, etc
    updateNowPlayingMetadata(metadata)

    // Update playback metadata, if available
    if let playbackMetadata = playbackMetadata(for: item) {
        updateNowPlayingPlaybackInfo(playbackMetadata)
    }
}

// Update playback state Now Playing info for significant events like rate changes and seeks
func handlePlaybackStateChanged(metadata: NowPlayableDynamicMetadata) {
    updateNowPlayingPlaybackInfo(metadata)
}
func updateNowPlayingPlaybackInfo(_ metadata: NowPlayabeDynamicMetadata) {
let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

nowPlayingInfo[MPMediaItemPropertyPlaybackDuration] = metadata.duration
nowPlayingInfo[MPNowPlayingInfoPropertyElapsedPlaybackTime] = metadata.position
nowPlayingInfo[MPNowPlayingInfoPropertyPlaybackRate] = metadata.rate

nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
func updateNowPlayingPlaybackInfo(_ metadata: NowPlayableDynamicMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    nowPlayingInfo[MPMediaItemPropertyPlaybackDuration] = metadata.duration
    nowPlayingInfo[MPNowPlayingInfoPropertyElapsedPlaybackTime] = metadata.position
    nowPlayingInfo[MPNowPlayingInfoPropertyPlaybackRate] = metadata.rate

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
func updateNowPlayingPlaybackInfo(_ metadata: NowPlayableDynamicMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()
    nowPlayingInfo[MPMediaItemPropertyPlaybackDuration] = metadata.duration
    nowPlayingInfo[MPNowPlayingInfoPropertyElapsedPlaybackTime] = metadata.position
    nowPlayingInfo[MPNowPlayingInfoPropertyPlaybackRate] = metadata.rate
    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}

func updateNowPlayingPlaybackInfo(_ metadata: NowPlayableDynamicMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    nowPlayingInfo[MPMediaItemPropertyPlaybackDuration] = metadata.duration
    nowPlayingInfo[MPNowPlayingInfoPropertyElapsedPlaybackTime] = metadata.position
    nowPlayingInfo[MPNowPlayingInfoPropertyPlaybackRate] = metadata.rate

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
Handle Skips and Seeks
Handle Skips and Seeks
Handle Skips and Seeks
Handle Skips and Seeks
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for play/pause/toggle commands ... */

    commandCenter.skipForwardCommand.preferredIntervals = [15.0]
    commandCenter.skipForwardCommand.addTarget {
        unowned self} event in
        guard let event = event as? MPSkipIntervalCommandEvent else { return .commandFailed }
        self.player.skipForward(by: event.interval)
        return .success
    }

    // Add handler for other skip and seek commands
    commandCenter.skipBackwardCommand.addTarget { /* ... */ }
    commandCenter.changePlaybackPositionCommand.addTarget { /* ... */ }
}
```swift
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* … register for play/pause/toggle commands … */

    commandCenter.skipForwardCommand.preferredIntervals = [15.0]
    commandCenter.skipForwardCommand.addTarget { [unowned self] event in
        guard let event = event as? MPSkipIntervalCommandEvent else { return .commandFailed }
        self.player.skipForward(by: event.interval)
        return .success
    }

    // Add handler for other skip and seek commands
    commandCenter.skipBackwardCommand.addTarget { /* … */ }
    commandCenter.changePlaybackPositionCommand.addTarget { /* … */ }
}
```
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* … register for play/pause/toggle commands … */

    commandCenter.skipForwardCommand.preferredIntervals = [15.0]
    commandCenter.skipForwardCommand.addTarget { [unowned self] event in
        guard let event = event as? MPSkipIntervalCommandEvent else { return .commandFailed }
        self.player.skipForward(by: event.interval)
        return .success
    }

    // Add handler for other skip and seek commands
    commandCenter.skipBackwardCommand.addTarget { /* … */ }
    commandCenter.changePlaybackPositionCommand.addTarget { /* … */ }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for play/pause/toggle commands ... */

    commandCenter.skipForwardCommand.preferredIntervals = [15.0]
    commandCenter.skipForwardCommand.addTarget { [unowned self] event in
        guard let event = event as? MPSkipIntervalCommandEvent else { return .commandFailed }
        self.player.skipForward(by: event.interval)
        return .success
    }

    // Add handler for other skip and seek commands
    commandCenter.skipBackwardCommand.addTarget { /* ... */ }
    commandCenter.changePlaybackPositionCommand.addTarget { /* ... */ }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* … register for play/pause/toggle commands … */

    commandCenter.skipForwardCommand.preferredIntervals = [15.0]
    commandCenter.skipForwardCommand.addTarget { [unowned self] event in
        guard let event = event as? MPSkipIntervalCommandEvent else { return .commandFailed }
        self.player.skipForward(by: event.interval)
        return .success
    }

    // Add handler for other skip and seek commands
    commandCenter.skipBackwardCommand.addTarget { /* … */ }
    commandCenter.changePlaybackPositionCommand.addTarget { /* … */ }
}

func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* … register for play/pause/toggle commands … */

    commandCenter.skipForwardCommand.preferredIntervals = [15.0]
    commandCenter.skipForwardCommand.addTarget {
        guard let event = event as? MPSkipIntervalCommandEvent else { return .commandFailed }
        self.player.skipForward(by: event.interval)
        return .success
    }

    // Add handler for other skip and seek commands
    commandCenter.skipBackwardCommand.addTarget { /* … */ }
    commandCenter.changePlaybackPositionCommand.addTarget { /* … */ }
}
Temporarily Disabling Commands

Use MPRemoteCommand’s isEnabled

Do not remove the target handler
Temporarily Disabling Commands

Use MPRemoteCommand’s `isEnabled`

Do not remove the target handler
Temporarily Disabling Commands

Use MPRemoteCommand’s `isEnabled`

Do not remove the target handler
Audio Languages and Subtitles
Audio Languages and Subtitles
Audio Languages and Subtitles
Audio Languages and Subtitles
Audio Languages and Subtitles

Audio
- English
- English AD
- French (Canada)

Subtitles
- Off
- Auto
- English CC
- Bulgarian
- Cantonese (Traditional)
- Chinese, Simplified
- Chinese, Traditional
- Czech
- Danish
- Dutch
Audio Languages and Subtitles

Provide audible and legible language options

Identify current selections

Handle option enable/disable commands
Audio Languages and Subtitles

Provide audible and legible language options

Identify current selections

Handle option enable/disable commands
Mapping AVAsset Media Selections

Load AVAsset’s audible and legible AVMediaSelectionGroup’s and options

Create a mapping between these and MPNowPlayingInfoLanguageOptionGroup

• Use convenience methods like makeNowPlayingInfoLanguageOptionGroup()
```swift
func updateNowPlayingPlaybackInfo(_ metadata: NowPlayableDynamicMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    /* ... set playback duration, time, rate, etc ... */

    nowPlayingInfo[MPNowPlayingInfoPropertyAvailableLanguageOptions] = metadata.optionGroups
    nowPlayingInfo[MPNowPlayingInfoPropertyCurrentLanguageOptions] = metadata.currentOptions

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
```
func updateNowPlayingPlaybackInfo(_ metadata: NowPlaybleDynamicMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()
    /* … set playback duration, time, rate, etc … */
    nowPlayingInfo[MPNowPlayingInfoPropertyAvailableLanguageOptions] = metadata.optionGroups
    nowPlayingInfo[MPNowPlayingInfoPropertyCurrentLanguageOptions] = metadata.currentOptions
    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
func updateNowPlayingPlaybackInfo(_ metadata: NowPlayableDynamicMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    /* … set playback duration, time, rate, etc … */

    nowPlayingInfo[MPNowPlayingInfoPropertyAvailableLanguageOptions] = metadata.optionGroups
    nowPlayingInfo[MPNowPlayingInfoPropertyCurrentLanguageOptions] = metadata.currentOptions

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}

func updateNowPlayingPlaybackInfo(_ metadata: NowPlayableDynamicMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    /* ... set playback duration, time, rate, etc ... */

    nowPlayingInfo[MPNowPlayingInfoPropertyAvailableLanguageOptions] = metadata.optionGroups
    nowPlayingInfo[MPNowPlayingInfoPropertyCurrentLanguageOptions] = metadata.currentOptions

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
func updateNowPlayingPlaybackInfo(_ metadata: NowPlayableDynamicMetadata) {
    let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()
    var nowPlayingInfo = nowPlayingInfoCenter.nowPlayingInfo ?? [String: Any]()

    /* ... set playback duration, time, rate, etc ... */

    nowPlayingInfo[MPNowPlayingInfoPropertyAvailableLanguageOptions] = metadata.optionGroups
    nowPlayingInfo[MPNowPlayingInfoPropertyCurrentLanguageOptions] = metadata.currentOptions

    nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
}
Audio Languages and Subtitles

Provide audible and legible language options
Identify current selections
Handle option enable/disable commands
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for other commands ... */

    commandCenter.enableLanguageOptionCommand.addTarget { [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }

        // Get the AVFoundation option & group to which the MediaPlayer language option references
        guard let (mediaSelectionOption, mediaSelectionGroup) = mediaSelectionOptionAndGroup(for: languageOption) else {
            return .commandFailed
        }

        currentItem.select(mediaSelectionOption, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* … register for other commands … */

    commandCenter.enableLanguageOptionCommand.addTarget {
        [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }

        // Get the AVFoundation option & group to which the MediaPlayer language option references
        guard let (mediaSelectionOption, mediaSelectionGroup) = mediaSelectionOptionAndGroup(for: languageOption) else {
            return .commandFailed
        }

        currentItem.select(mediaSelectionOption, in: mediaSelectionGroup)
        handlePlaybackChange()

        return .success
    }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for other commands ... */

    commandCenter.enableLanguageOptionCommand.addTarget {
        [unowned self] event in
            guard let event = event as? MPChangeLanguageOptionCommandEvent else {
                return .commandFailed
            }
            guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }

            // Get the AVFoundation option & group to which the MediaPlayer language option references
            guard let (mediaSelectionOption, mediaSelectionGroup) = mediaSelectionOptionAndGroup(for: languageOption) else { return .commandFailed }

            currentItem.select(mediaSelectionOption, in: mediaSelectionGroup)
            handlePlaybackChange()
            return .success
        }
    }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for other commands ... */

    commandCenter.enableLanguageOptionCommand.addTarget { [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }
        // Get the AVFoundation option & group to which the MediaPlayer language option references
        guard let (mediaSelectionOption, mediaSelectionGroup) =
            mediaSelectionOptionAndGroup(for: languageOption) else {
                return .commandFailed
            }

        currentItem.select(mediaSelectionOption, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for other commands ... */

    commandCenter.enableLanguageOptionCommand.addTarget {
        [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }
        // Get the AVFoundation option & group to which the MediaPlayer language option references
        guard let (mediaSelectionOption, mediaSelectionGroup) = mediaSelectionOptionAndGroup(for: languageOption) else { return .commandFailed }
        currentItem.select(mediaSelectionOption, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}

func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for other commands ... */

    commandCenter.enableLanguageOptionCommand.addTarget { [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }

        // Get the AVFoundation option & group to which the MediaPlayer language option references
        guard let (mediaSelectionOption, mediaSelectionGroup) = mediaSelectionOptionAndGroup(for: languageOption) else {
            return .commandFailed
        }

        currentItem.select(mediaSelectionOption, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for other commands ... */
    commandCenter.enableLanguageOptionCommand.addTarget { /* ... */ }
    commandCenter.disableLanguageOptionCommand.addTarget {
        [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }

        // Get the AVFoundation group to which the MediaPlayer language option references
        guard let group = mediaSelectionGroup(for: languageOption) else { return .commandFailed }

        guard mediaSelectionGroup.allowsEmptySelection else { return .commandFailed }
        currentItem.select(nil, in: mediaSelectionGroup)

        handlePlaybackChange()
        return .success
    }
}

func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for other commands ... */
    commandCenter.enableLanguageOptionCommand.addTarget {
        /* ... */
    }
    commandCenter.disableLanguageOptionCommand.addTarget {
        [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }
        // Get the AVFoundation group to which the MediaPlayer language option references
        guard let group = mediaSelectionGroup(for: languageOption) else { return .commandFailed }
        guard mediaSelectionGroup.allowsEmptySelection else { return .commandFailed }
        currentItem.select(nil, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* ... register for other commands ... */
    commandCenter.enableLanguageOptionCommand.addTarget { /* ... */ }
    commandCenter.disableLanguageOptionCommand.addTarget { [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else {
            return .noActionableNowPlayingItem
        }
        // Get the AVFoundation group to which the MediaPlayer language option references
        guard let group = mediaSelectionGroup(for: languageOption) else {
            return .commandFailed
        }
        guard mediaSelectionGroup.allowsEmptySelection else {
            return .commandFailed
        }
        currentItem.select(nil, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* … register for other commands … */
    commandCenter.enableLanguageOptionCommand.addTarget { /* … */ }
    commandCenter.disableLanguageOptionCommand.addTarget {
        [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }

        // Get the AVFoundation group to which the MediaPlayer language option references
        guard let group = mediaSelectionGroup(for: languageOption) else { return .commandFailed }
        guard mediaSelectionGroup.allowsEmptySelection else { return .commandFailed }
        currentItem.select(nil, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}

func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* … register for other commands … */
    commandCenter.enableLanguageOptionCommand.addTarget { /* … */ }
    commandCenter.disableLanguageOptionCommand.addTarget { [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }

        // Get the AVFoundation group to which the MediaPlayer language option references
        guard let group = mediaSelectionGroup(for: languageOption) else { return .commandFailed }
        guard mediaSelectionGroup.allowsEmptySelection else { return .commandFailed }

        currentItem.select(nil, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}
func setupRemoteTransportControls() {
    let commandCenter = MPRemoteCommandCenter.shared()
    /* … register for other commands … */
    commandCenter.enableLanguageOptionCommand.addTarget { /* … */ }
    commandCenter.disableLanguageOptionCommand.addTarget { [unowned self] event in
        guard let event = event as? MPChangeLanguageOptionCommandEvent else {
            return .commandFailed
        }
        guard let currentItem = player.currentItem else { return .noActionableNowPlayingItem }

        // Get the AVFoundation group to which the MediaPlayer language option references
        guard let group = mediaSelectionGroup(for: languageOption) else { return .commandFailed }

        guard mediaSelectionGroup.allowsEmptySelection else { return .commandFailed }
        currentItem.select(nil, in: mediaSelectionGroup)
        handlePlaybackChange()
        return .success
    }
}
When to Register for Remote Controls

Register:
• User selected content

Deregister:
• No content available to control
• Clear out `nowPlayingInfo` when you no longer have an item to display
tvOS and Remote Controls
Best quality video
AirPlay picker
Remote controls
AirPlay multitasking
Long-form video apps
AirPlay Multitasking

Keep your content playing

Don’t accidentally interrupt other apps
AirPlay Multitasking

Keep your content playing

Don’t accidentally interrupt other apps
AirPlay Multitasking

This video is playing on "Living Room"
AirPlay Multitasking
Keep Content Playing in the Background

Set `UIBackgroundMode` to include audio

Configure `AVAudioSession`
Audio Session Handling for Video Apps

Use .playback category
Audio Session Handling for Video Apps

Use `.playback` category

Go active when user presses “play” button
Audio Session Handling for Video Apps

Use `.playback` category

Go active when user presses “play” button

Stay active, unless interrupted
Audio Session Handling for Video Apps

Use `.playback` category

Go active when user presses “play” button

Stay active, unless interrupted

Handle interruptions

• Begin — Update UI and internal state
• End — Honor `AVAudioSessionInterruptionOptionShouldResume`
AirPlay Multitasking

Keep your content playing

Don’t accidentally interrupt other apps
Do Not Accidentally Interrupt Others

Some videos should never AirPlay
- App launch videos
- Auto-playing visual flourish

Set `AVPlayer.allowsExternalPlayback = false`
Do Not Accidentally Interrupt Others

Some videos should never AirPlay
• App launch videos
• Auto-playing visual flourish

Set AVPlayer.allowsExternalPlayback = false
Do Not Accidentally Interrupt Others

Do not auto-play primary content

• User should always be in control of when media starts
AirPlay to Living Room
“WWDC” will play on this TV automatically.

Videos
Favorites
Downloaded
All Videos

Continue Watching

Designing Fluid Interfaces
WWDC 2018

Building Apps for watchOS
AirPlay to Living Room

“MyApp” will play on this TV automatically.
Best quality video
AirPlay picker
Remote controls
AirPlay multitasking
Long-form video apps
Long-Form Video Apps

Marty Pye, AVKit
What is a Long-Form Video App?

- Movies
- TV

- Social Media
- News Clips
How is AirPlay Routing different for long-form video apps?
Check out this dirt bike video!
Where do you want to play this video?
You can use AirPlay to watch this using the "Living Room" TV.

- This iPhone
- Living Room
- Cancel

Mark as Watched
How to Take Advantage of These New Features?
How to Take Advantage of These New Features?

Register your app as a long-form video app
Registering as a Long-Form Video App

Add this key to your app’s Info.plist:

```xml
<key>AVInitialRouteSharingPolicy</key>
<string>LongFormVideo</string>
```
How to Take Advantage of These New Features?

• Register your app as a long-form video app
• Prepare the routing system prior to playback
Prepare Routing System Prior to Playback

App
- Browse

Movie
  - Play

System
  - AirPlay Routing System
Prepare Routing System Prior to Playback

App

Browse

Movie

System

AirPlay Routing System

prepare

Play
Prepare Routing System Prior to Playback

```swift
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(completionHandler: {
    (shouldStartPlayback, routeSelection) in
        // Handle completion.
})
```
Prepare Routing System Prior to Playback

```swift
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(
    completionHandler: { (shouldStartPlayback, routeSelection) in
        // Handle completion.
    }
)
```
Prepare Routing System Prior to Playback

```swift
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(completionHandler:
{ (shouldStartPlayback, routeSelection) in
   // Handle completion.
}
```
Where do you want to play this video?
You can use AirPlay to watch this using the "Living Room" TV.

- This iPhone
- Living Room
- Cancel

Mark as Watched

Videos  Schedule  News  Venue
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(completionHandler: {
    (shouldStartPlayback, routeSelection) in
    if shouldStartPlayback {
        switch routeSelection {
        case .local:
            // Present UI streamlined for local playback.
        case .external:
            // Present UI streamlined for external playback.
        case .none:
            // Playback cancelled.
        }
    }
})
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(completionHandler: { (shouldStartPlayback, routeSelection) in
    if shouldStartPlayback {
        switch routeSelection {
        case .local:
            // Present UI streamlined for local playback.
        case .external:
            // Present UI streamlined for external playback.
        case .none:
            // Playback cancelled.
        }
    }
})
Where do you want to play this video?

You can use AirPlay to watch this using the "Living Room" TV.

- This iPhone
- Living Room
- Cancel

Mark as Watched

Videos Schedule News Venue
Where do you want to play this video?
You can use AirPlay to watch this using the "Living Room" TV.

This iPhone
Living Room
Cancel

Mark as Watched
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(completionHandler: {(shouldStartPlayback, routeSelection) in
    if shouldStartPlayback {
        switch routeSelection {
        case .local:
            // Present UI streamlined for local playback.
        case .external:
            // Present UI streamlined for external playback.
        case .none:
            // Playback cancelled.
        }
    }
}}
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(completionHandler:
{
    (shouldStartPlayback, routeSelection) in
        if shouldStartPlayback {
            switch routeSelection {
                case .local:
                    // Present UI streamlined for local playback.
                case .external:
                    // Present UI streamlined for external playback.
                case .none:
                    // Playback cancelled.
            }
        }
    }
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(completionHandler: { (shouldStartPlayback, routeSelection) in
    if shouldStartPlayback {
        switch routeSelection {
        case .local:
            // Present UI streamlined for local playback.
        case .external:
            // Present UI streamlined for external playback.
        case .none:
            // Playback cancelled.
        }
    }
})
AVAudioSession.sharedInstance().prepareRouteSelectionForPlayback(completionHandler: {
    (shouldStartPlayback, routeSelection) in
    if shouldStartPlayback {
        switch routeSelection {
        case .local:
            // Present UI streamlined for local playback.
        case .external:
            // Present UI streamlined for external playback.
        case .none:
            // Playback cancelled.
        }
    }
}}
Testing AirPlay Suggestions
Testing AirPlay Suggestions
Testing AirPlay Suggestions

SUGGESTION TEST MODE

- Default
- Always Prompt User with Suggested TV
- Always Use Suggested TV

When using a mode other than “Default”, select an AirPlay-capable TV as the suggested TV to use for testing. This TV must be connected to the same Wi-Fi network as this iPhone.
Testing AirPlay Suggestions

- Default
- Always Prompt User with Suggested TV
- Always Use Suggested TV

When using a mode other than “Default”, select an AirPlay-capable TV as the suggested TV to use for testing. This TV must be connected to the same Wi-Fi network as this iPhone.

- Choose a Suggested TV
- Bedroom
- Living Room
Testing AirPlay Suggestions
Testing AirPlay Suggestions
You can still manually AirPlay!
Sample Code

LongFormVideoApp
Summary

AirPlay Picker
Remote Controls
AirPlay Multitasking
Long-Form Video Apps
Routing System
More Information

developer.apple.com/wwdc19/501

<table>
<thead>
<tr>
<th>AirPlay Lab</th>
<th>Tuesday, 12:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivering Intuitive Media Playback with AVKit</td>
<td>Thursday, 10:00</td>
</tr>
</tbody>
</table>